The minimum cost spanning tree problem

Giovanni Righini

UNIVERSITÀ DEGLI STUDI DI MILANO

Definitions - 1

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a tree if and only if it is *connected* and *acyclic*.

- Connectivity: for each cut, at least one edge belongs to the tree.
- Acyclicity: for each cycle, at least one edge does not belong to the tree.

Given a $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ a subset $\mathcal{F} \subseteq \mathcal{E}$ is:

- a forest if it does not contain cycles;
- a connector if $(\mathcal{V}, \mathcal{F})$ is connected;
- a spanning tree if $(\mathcal{V}, \mathcal{F})$ is a tree;
- a maximal forest if there is no forest containing it.

Definitions - 2

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ has a spanning tree if and only if it is connected.

Given a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{F}$ is a spanning tree if and only if:

- $\mathcal{F}$ is a maximal forest;
- $\mathcal{F}$ is a minimal connector;
- $\mathcal{F}$ is a forest with $|\mathcal{F}| = |\mathcal{V}| - 1$ edges;
- $\mathcal{F}$ is a connector with $|\mathcal{F}| = |\mathcal{V}| - 1$ edges.

Definitions - 3

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with k connected components, every maximal forest in it has $|\mathcal{V}| - k$ edges.

It forms a spanning tree in each connected component of $\mathcal{G}$.

Every maximal forest is also a maximum cardinality forest.

Analogously, every connector contains a minimum cardinality connector.

The Minimum Spanning Tree Problem

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected graph.

Let $c : \mathcal{E} \to \Re$ a cost function.

We define $\forall \mathcal{F} \subseteq \mathcal{E}$:

$$c(\mathcal{F}) = \sum_{e \in \mathcal{F}} c_e$$

Problem (Minimum Spanning Tree Problem). Find a spanning tree of minimum cost in $\mathcal{G}$.

Properties of spanning trees (1)

Property 1. A spanning tree of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ has $|\mathcal{V}| - 1$ edges.

Owing to Property 1, we can assume $c_e \geq 0$ for all edges $e \in \mathcal{E}$.

If not, we can add a “large enough” constant to all edge costs. This does not change the ranking of the feasible solutions, because all feasible solutions contain the same number of edges.

Properties of spanning trees

Property 2. If T is a spanning tree containing edge e, then $T \setminus \{e\}$ is a forest made of two connected components, separated by a unique cut $C(T, e)$.

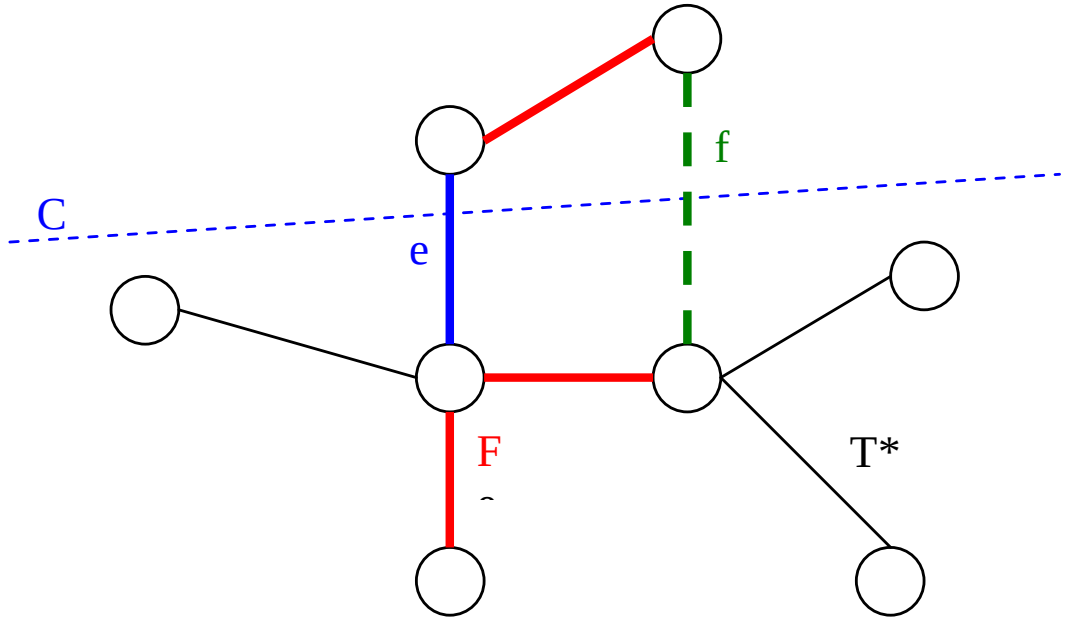
Property 3. If T is a spanning tree and u and v are two vertices of the graph, T contains a unique path $P(T, u, v)$ between them.

Definition. $\mathcal{F}$ is a good forest iff it belongs to a minimum cost spanning tree.

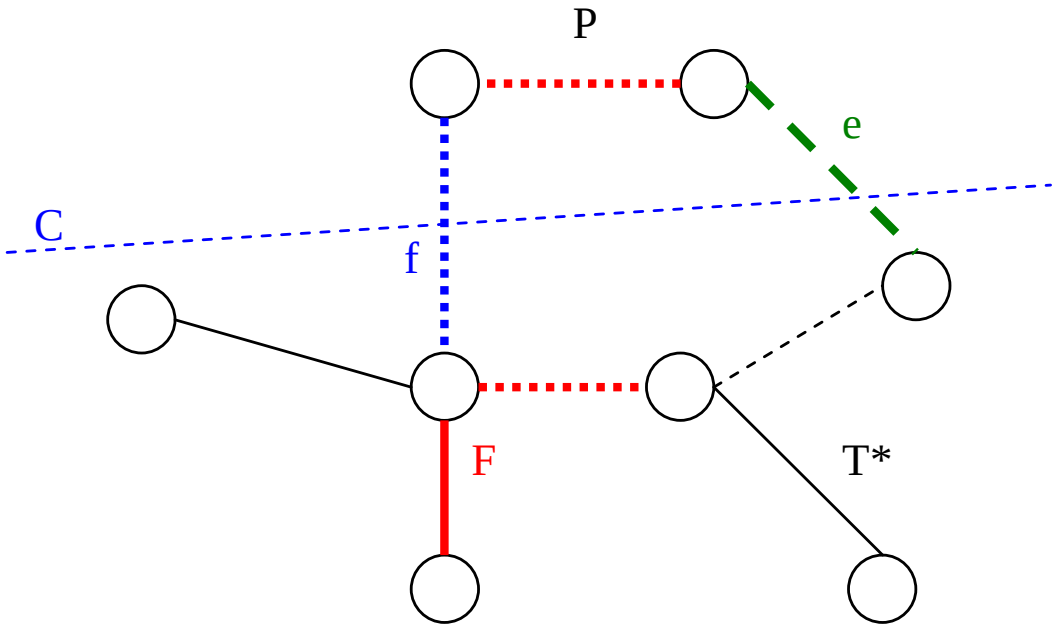
Theorem. Given a good forest $\mathcal{F}$ and given an edge $e \notin \mathcal{F}$, $\mathcal{F} \cup \{e\}$ is a good forest iff there is a cut C disjoint from $\mathcal{F}$ such that e is an edge with minimum cost in C.

Proof

Necessity. Let T^* be a minimum cost spanning tree containing $\mathcal{F} \cup \{e\}$. Then both $\mathcal{F}$ and $\mathcal{F} \cup \{e\}$ are good forests.
Let $C(T^*, e)$ be the cut disjoint from $T^* \setminus \{e\}$. Then $C(T^*, e)$ is also disjoint from $\mathcal{F}$.
Consider any $f \in C(T^*, e)$: $T' = T^* \setminus \{e\} \cup \{f\}$ is a spanning tree.
Since T^* is optimal, then $c(T^*) \leq c(T')$ and then $c(e) \leq c(f)$.
Then, e is an edge of minimum cost in $C(T^*, e)$.



Sufficiency. Let T^* be a min cost spanning tree containing $\mathcal{F}$; then $\mathcal{F}$ is good. Consider a cut C disjoint from $\mathcal{F}$ and e of min cost in C.
If $e \in T^*$ then $\mathcal{F} \cup \{e\}$ is good.
If $e = [u, v] \notin T^*$, then let $P(T^*, u, v)$ be the path in T^* between u and v: $P(T^*, u, v)$ contains at least one edge $f \in C$. Then,
$T' = T^* \setminus \{f\} \cup \{e\}$ is a spanning tree.
Since $c(e) \leq c(f)$, then $c(T') \leq c(T^*)$ and T' is a min cost spanning tree.
Since $\mathcal{F} \cup \{e\}$ is contained in T', it is good.



Algorithms

Almost all MSTP algorithms exploit the previous theorem: start with an empty (good) forest $\mathcal{F}$ and extend it iteratively with an edge satisfying the theorem requirement, i.e. an edge of minimum cost in a cut C disjoint from $\mathcal{F}$.

Different algorithms are obtained by different choices of C. The two most common algorithms are:

- Jarnik (1930), Kruskal (1956), Prim (1957), Dijkstra (1959): C is the cut that separates the connected component including a predefined vertex;
- Kruskal (1956), Loberman e Weinberger (1957), Prim (1957): C is the cut that separates the two connected components including the endpoints of the minimum cost edge e in a sorted list.

Prim algorithm: data-structures

Data-structures of Prim algorithm:

- T: current forest (edge set)
- z: cost of T
- $flag[v]$: binary flag indicating whether vertex v is spanned by T
- r: special vertex, arbitrarily chosen
- $cost[v]$: minimum cost among the edges connecting v with vertices in T
- $pred[v]$: the other endpoint of an edge of minimum cost between v and T

Prim algorithm: initialization

```
// Procedure Init
T ← ∅
z ← 0
for v = 1, ..., n do
    flag[v] ← 0
flag[r] ← 1
for v = 1, ..., n do
    cost[v] ← c[r, v]
    pred[v] ← r
```

Prim algorithm

```
// Prim algorithm
Init
for k = 1, ..., n − 1 do
    mincost ← ∞
    for v = 1, ..., n do
        if (flag[v] = 0) ∧ (cost[v] < mincost) then
            v̄ ← v
            mincost ← cost[v]
    T ← T ∪ {[pred[v̄], v̄]}
    z ← z + mincost
    flag[v̄] ← 1
    for v = 1, ..., n do
        if (flag[v] = 0) ∧ (c[v̄, v] < cost[v]) then
            pred[v] ← v̄
            cost[v] ← c[v̄, v]
```

Prim algorithm

The complexity is $O(n^2)$.

With 2-heaps it is possible to obtain $O(m \log n)$.

With Fibonacci heaps it is possible to obtain $O(m + n \log n)$.

Kruskal algorithm (1956)

```
// Kruskal algorithm
T ← ∅; z ← 0
E ← Sort(ℰ)
for v = 1, ..., n do
    List[v] ← {v}; head[v] ← v; card[v] ← 1
while (|T| < n − 1) do
    [u, v] ← argmin_{e∈E}{c_e}; E ← E\{[u, v]}
    if (head[u] ≠ head[v]) then
        T ← T ∪ {[u, v]}; z ← z + c[u, v]
        if (card[v] > card[u]) then
            Swap(u, v)
        L[u] ← L[u] ∪ L[v]; card[u] ← card[u] + card[v]
        for i ∈ L[v] do
            head[i] ← head[u]
```

Kruskal algorithm: complexity

Sorting the edges requires $O(m \log n)$.

The edges can be partitioned into n subsets of cardinality $k_i \; \forall i = 1, \ldots, n$ by arbitrary selecting one of their endpoints.

The following properties hold:

- $k_i \leq n \; \forall i = 1, \ldots, n$
- $\sum_{i=1}^{n} k_i = m$

Each subset can be sorted in $O(k_i \log k_i)$.

Hence the overall complexity is $O(\sum_{i=1}^{n} k_i \log k_i)$.

For the first property above, $\sum_{i=1}^{n} k_i \log k_i \leq \sum_{i=1}^{n} k_i \log n$.

For the second property above, $\sum_{i=1}^{n} k_i \log n = m \log n$.

Kruskal algorithm: complexity

Once n sorted lists have been produced, they can be merged in a unique sorted list in $O(m \log n)$.

In $O(n)$ a binary heap containing the (partially sorted) heads of the n lists is built.

For m times the root is extracted from the heap and the heap is rearranged in $O(\log n)$: this takes $O(m \log n)$.

Kruskal algorithm: complexity

After sorting, the complexity is $O(m + n \log n)$ and it can be obtained with linked lists.

A list L is associated with each component of the current forest.
For each vertex $v \in V$, $head(v)$ is the head of the list L_v containing v.
Initially $head(v)$:=v and L_v:=$\{v\}$ for all vertices.

At each iteration:

- test whether the next edge $e = [u, v]$ in the list would close a cycle or not;
- if not, update the data-structure.

Kruskal algorithm: complexity

The test is: $head(u) = head(v)$?
It is executed in $O(1)$ at most m times.
Hence it requires $O(m)$ overall.

When extending the current forest with $e = [u, v]$:

- detectable the shortest list among L_u and L_v (in $O(1)$) with a cardinality counter associated with each list;
- append it to the longest one (in $O(1)$);
- update the cardinality of the longest list (in $O(1)$);
- update the head for all vertices in the shortest list.

Property. No vertex can belong to the shortest list more than $\log n$ times, because the size of the shortest list at least doubles every time its head is updated.

Therefore the head update operation requires at most $O(\log n)$ for each vertex, i.e. $O(n \log n)$ overall.

Boruvka algorithm (1926)

It requires that all edge costs are different from each other and it allows for a parallel implementation.

// Boruvka algorithm
$\mathcal{F} \leftarrow \emptyset$
while $(|\mathcal{F}| < n - 1)$ **do**
  $\mathcal{F}' \leftarrow \emptyset$
  for $K \in$ Components$(\mathcal{F})$ **do**
    $\mathcal{F}' \leftarrow \mathcal{F}' \cup \{\text{argmin}_{e \in \delta(K)}\{c_e\}\}$
  $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}'$

Let $e_1, e_2, \ldots, e_k$ the edges inserted into $\mathcal{F}'$ at a generic iteration with $c_{e_1} < c_{e_2} < \ldots < c_{e_k}$.
For each $i = 1, \ldots, k$, e_i is the minimum cost edge leaving $\mathcal{F} \cup \{e_1, e_2, \ldots, e_{i-1}\}$, because none of $\{e_1, e_2, \ldots, e_{i-1}\}$ leaves the i^{th} component.
Then $\mathcal{F}$ is a good forest as if the edges were inserted sequentially.

Dual greedy algorithm

It deletes edges from a connector instead of inserting them into a forest.

Definition. A connector is good if it contains at least one minimum cost spanning tree.

Theorem. Given a good connector K and an edge $e \in K$, $K \setminus \{e\}$ is a good connector iff K contains a cycle C such that e is the maximum cost edge in C.

Kruskal (1956): sort the edges and starting from the connector $K = \mathcal{E}$, iteratively delete the maximum cost edge which is not a bridge (i.e. without disconnecting the graph).

More algorithms

Dijkstra algorithm (1960):
Arbitrarily sort the edges and iteratively insert them into a forest (initially empty).
When an insertion forms a cycle C, delete the maximum cost edge in C.

Kalaba algorithm (1960):
The same, but starting from an arbitrary spanning tree.

A mathematical programming model

$$\min z = \sum_{e \in \mathcal{E}} c_e x_e$$

$$\text{s.t.} \sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subset \mathcal{V} \qquad (1)$$

$$x_e \in \{0, 1\} \qquad \forall e \in \mathcal{E}$$

A cut $\delta(S)$ is the subset of edges with one endpoint in S.

Constraints 1 impose connectivity. Acyclicity comes for free from cost minimization.

Another mathematical programming model

$$\min z = \sum_{e \in \mathcal{E}} c_e x_e$$

$$\text{s.t.} \sum_{e \in \mathcal{E}} x_e = n - 1 \qquad (2)$$

$$\sum_{e \in \mathcal{E}(S)} x_e \leq |S| - 1 \qquad \forall S \subset \mathcal{V} \qquad (3)$$

$$x_e \in \{0, 1\} \qquad \forall e \in \mathcal{E}$$

$\mathcal{E}(S)$ is the edge subset of the subgraph induced by S.

Constraints (3) impose acyclicity instead of connectivity. Hence we also need constraint (2) to impose connectivity.

Integrality conditions are redundant.

The maximum value x_e can take is 1, because of the acyclicity constraint with $|S| = 2$;

The dual model

$$\min z = \sum_{e \in \mathcal{E}} c_e x_e$$

$$\text{s.t.} \sum_{e \in \mathcal{E}} x_e = n - 1$$

$$-\sum_{e \in \mathcal{E}(S)} x_e \geq -(|S| - 1) \qquad \forall S \subset \mathcal{V}$$

$$x_e \geq 0 \qquad \forall e \in \mathcal{E}$$

This linear program has a dual:

$$\max w = -\sum_{S \subset \mathcal{V}} (|S| - 1) y_S + (n-1) y_V$$

$$\text{s.t.} \; -\sum_{S \subset \mathcal{V} : e \in \mathcal{E}(S)} y_S + y_V \leq c_e \qquad \forall e \in \mathcal{E}$$

$$y_S \geq 0 \qquad \forall S \subset \mathcal{V}$$

y_V free

Complementary slackness conditions

Primal C.S.C.:

$$x_e(c_e + \sum_{S \subset \mathcal{V}: e \in \mathcal{E}(S)} y_S - y_V) = 0 \quad \forall e \in \mathcal{E}.$$

Dual C.S.C.:

$$y_V(\sum_{e \in \mathcal{E}} x_e - (n-1)) = 0.$$

$$y_S(|S| - 1 - \sum_{e \in \mathcal{E}(S)} x_e) = 0 \quad \forall S \subset \mathcal{V}.$$

The initial primal solution $x_e = 0 \; \forall e \in \mathcal{E}$ is primal infeasible (and super-optimal): the cardinality constraint is violated.
The initial dual solution $y_S = y_V = 0 \; \forall S \subset \mathcal{V}$ is dual feasible (and sub-optimal).

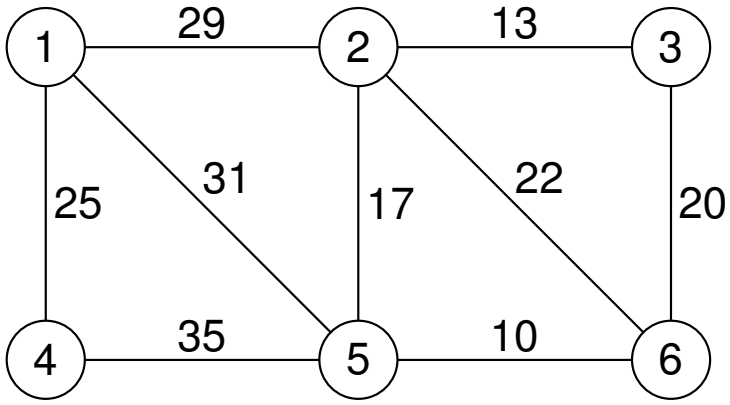
Primal-dual algorithm

Primal-dual interpretation of Kruskal algorithm:

- Dual iteration.
  Acyclicity constraints are always kept satisfied.
  The only violated primal constraint is $\sum_{e \in \mathcal{E}} x_e = n - 1$.
  The corresponding dual variable y_V is increased (dual ascent), until a dual constraint becomes active; it corresponds to a primal variable x_e.
- Primal iteration.
  The primal variable x_e enters the basis and the infeasibility of the cardinality constraint decreases.
  Some acyclicity constraints become active: hence some dual variables can enter the dual basis.
  In turn, this allows to increase y_V further.

The CSC $y_V(\sum_{e \in \mathcal{E}} x_e - (n-1)) = 0$ corresponds to a violated equality constraint in the primal problem.

Kruskal algorithm: an example

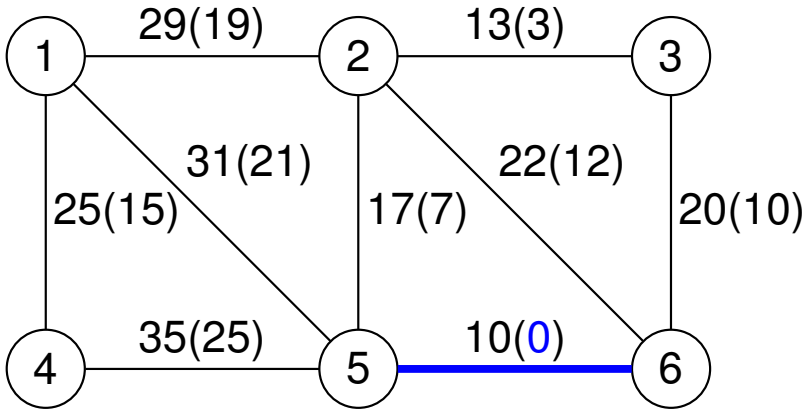


$k = 0$.
$x = 0$.
$z = 0$.

2 3 5 6 1 4

$y_V = 0$.
$w = 0$.
Dual C.S.C.:
$y_V(k - (n-1)) = 0(0-5) = 0$.

An example: dual iteration 1

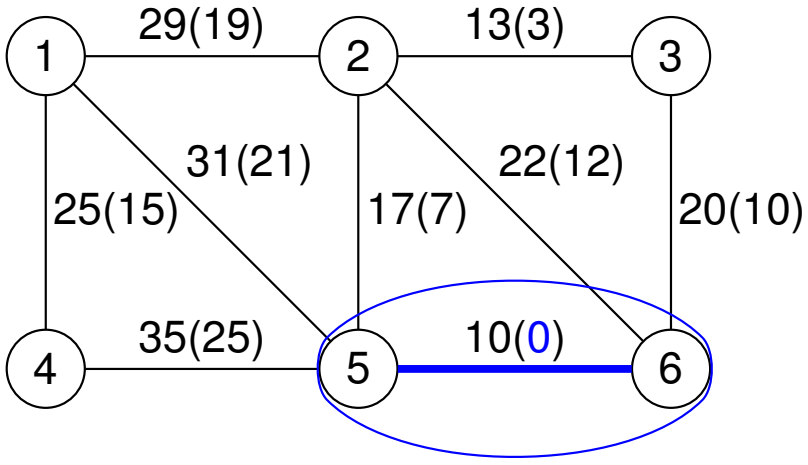


$k = 0$.
$x = 0$.
$z = 0$.

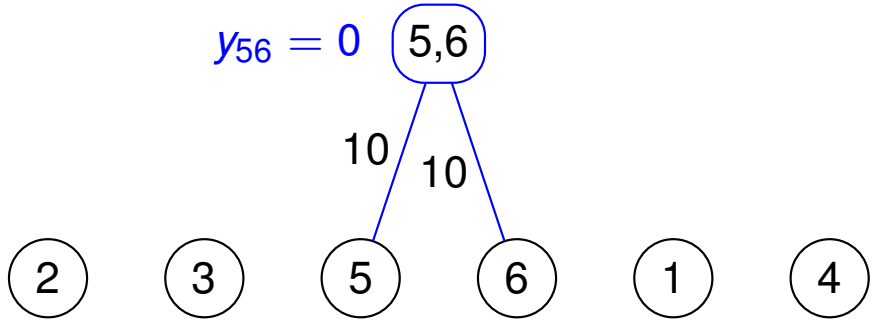
2 3 5 6 1 4

$y_V = 10$.
$w = 10 \times 5 = 50$.
Dual C.S.C.:
$y_V(k-(n-1)) = 10(0-5) = -50$.

An example: primal iteration 1

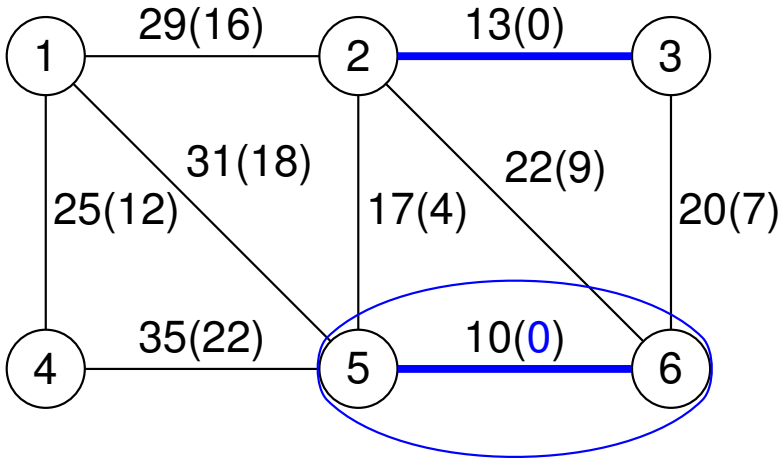


$k = 1$.
$x_{56} = 1$.
$z = 10$.

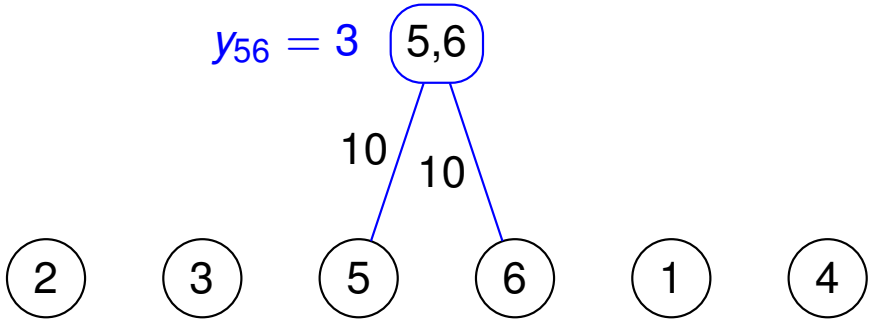


$y_V = 10$.
$w = -0 \times 1 + 10 \times 5 = 50$.
Dual C.S.C.:
$y_V(k-(n-1)) = 10(1-5) = -40$.

An example: dual iteration 2

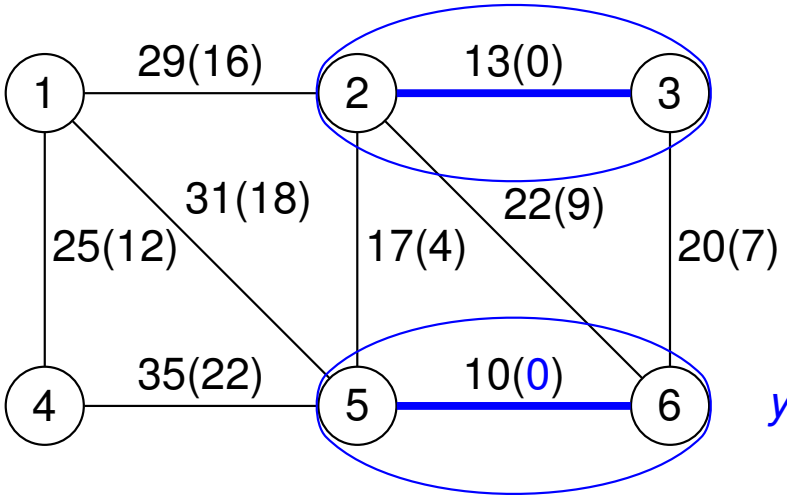


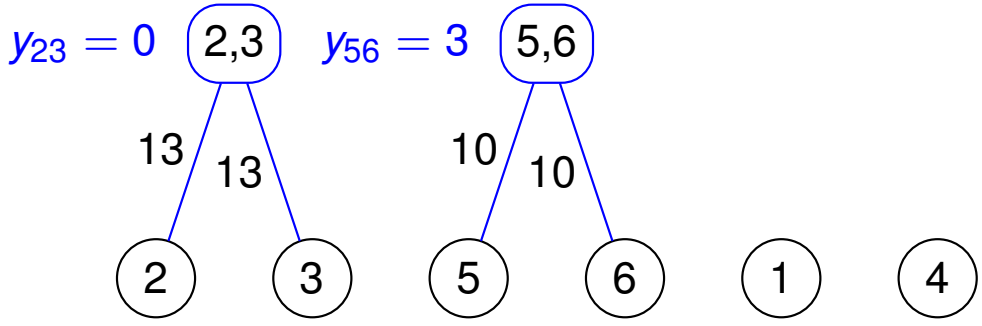
$k = 1$.
$x_{56} = 1$.
$z = 10$.



$y_V = 13$.
$w = -3 \times 1 + 13 \times 5 = 62$.
Dual C.S.C.:
$y_V(k-(n-1)) = 13(1-5) = -52$.

An example: primal iteration 2

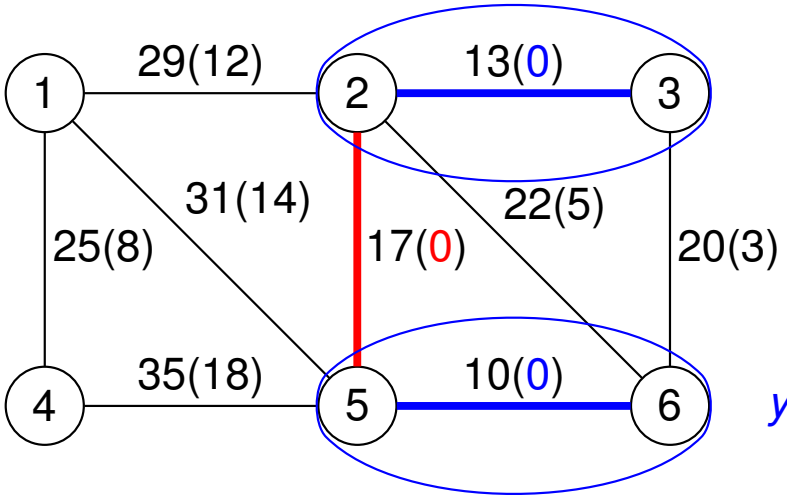


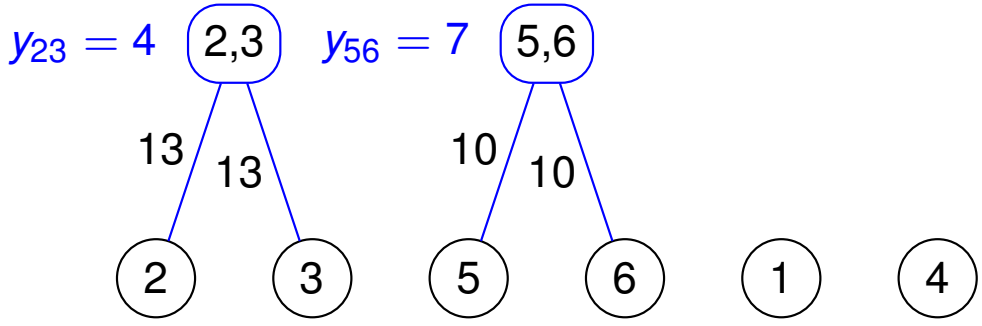


$k = 2$.
$x_{56} = x_{23} = 1$.
$z = 23$.

$y_V = 13$.
$w = -3 \times 1 - 0 \times 1 +$
$+13 \times 5 = 62$.
Dual C.S.C.:
$y_V(k - (n-1)) = 13(2-5) = -39$.

An example: dual iteration 3

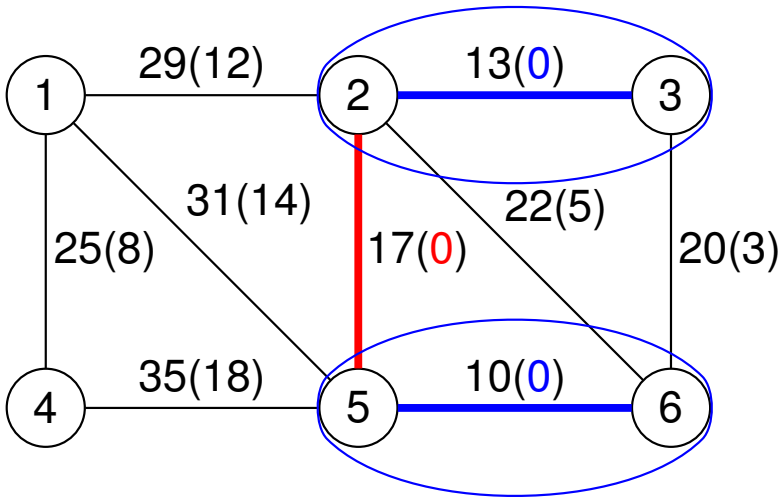




$k = 2$.
$x_{56} = x_{23} = 1$.
$z = 23$.

$y_V = 17$.
$w = -7 \times 1 - 4 \times 1 +$
$+17 \times 5 = 74$.
Dual C.S.C.:
$y_V(k - (n-1)) = 17(2-5) = -51$.

An observation



$k = 3$.
$x_{56} = x_{23} = x_{25} = 1$.
$z = 40$.

Several primal constraints are now active: several dual variables can enter the basis:

$$\{2,5\}: \quad x_{25} = 1$$
$$\{2,3,5\}: \quad x_{23} + x_{25} = 2$$
$$\{2,5,6\}: \quad x_{25} + x_{56} = 2$$
$$\{2,3,5,6\}: \quad x_{23} + x_{25} + x_{56} = 3$$

Which dual variable should we choose to enter the basis?

An observation

Active dual constraints:

$$
\begin{aligned}
[2,5] &: y_{25} + y_{235} + y_{256} + y_{2356} + c_{25} = y_V \\
[2,3] &: y_{23} + y_{235} \qquad\quad + y_{2356} + c_{23} = y_V \\
[5,6] &: y_{56} \qquad\quad + y_{256} + y_{2356} + c_{56} = y_V
\end{aligned}
$$

For each unit increase of y_V (providing value 5 in the dual objective), we can keep the dual constraints active with a unit increase of:

- y_{56} and y_{23} and y_{25}: the cost is 1+1+1=3;
- y_{56} and y_{235}: the cost is 1+2=3;
- y_{23} and y_{256}: the cost is 1+2=3;
- y_{2356}: the cost is 3.

From the viewpoint of the dual active constraints and the dual objective function, all these possibilities are equivalent.

An observation

But from the viewpoint of non-active dual constraints, they are not: if we choose y_{2356} to enter the basis, no dual constraint corresponding to any other edge in $\{2, 3, 5, 6\}$ will become active.

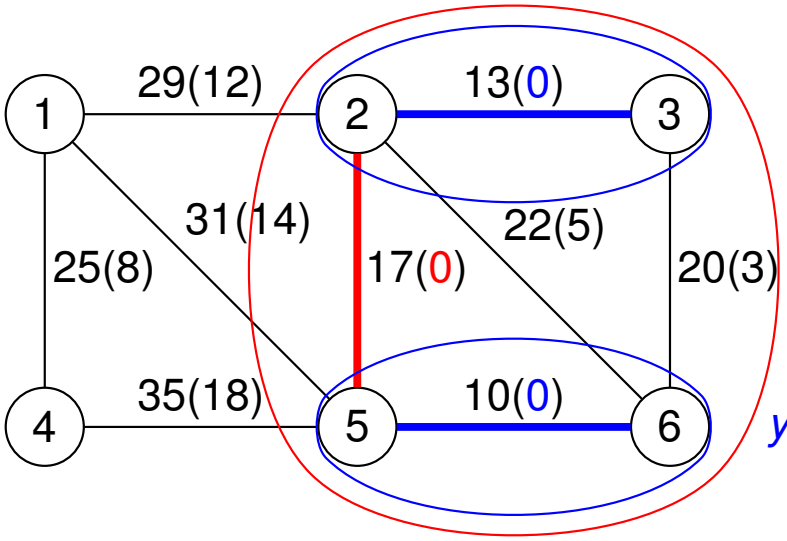
Edges reduced costs are:

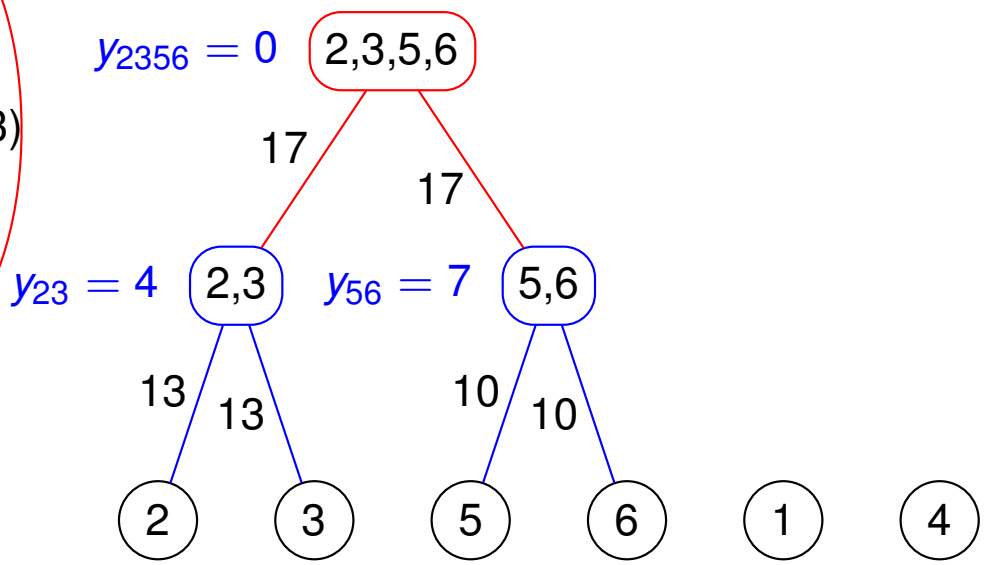
$$\overline{c}_e = c_e + \sum_{S \subset V : e \in \mathcal{E}(S)} y_S - y_V.$$

Reduced costs of all edges "covered" by basic dual variables do not decrease anymore.

The dual variable y_{2356} dominates the others; its corresponding primal constraint $\sum_{e \in \mathcal{E}(\{2,3,5,6\})} x_e \leq 3$ dominates those corresponding to the others: i.e., if we do not select more edges in $S = \{2, 3, 5, 6\}$, we cannot select more edges in any subset of S.

An example: primal iteration 3





$k = 3$.

$x_{56} = x_{23} = x_{25} = 1$.

$z = 40$.

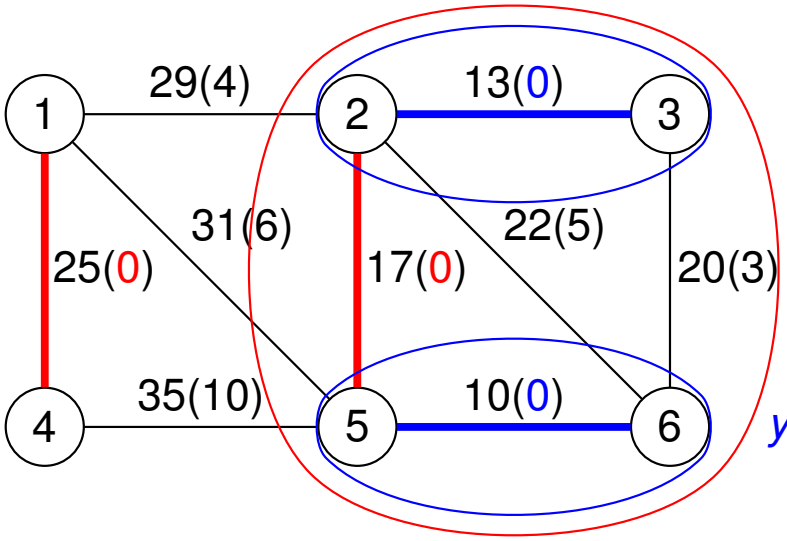
$y_V = 17$.

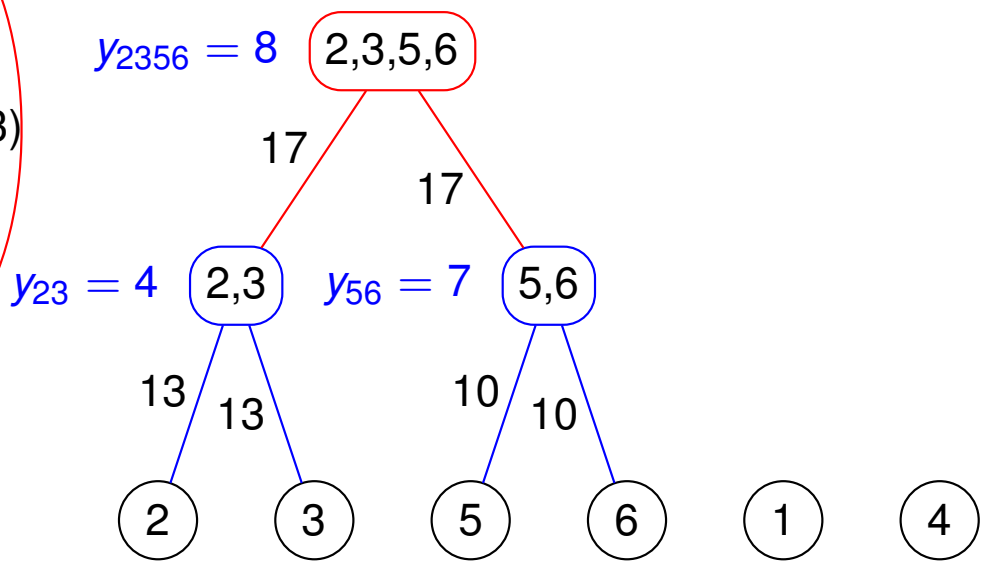
$w = -7 \times 1 - 4 \times 1 + +17 \times 5 = 74$.

Dual C.S.C.:

$y_V(k - (n-1)) = 17(3-5) = -34$.

An example: dual iteration 4





$k = 3$.

$x_{56} = x_{23} = x_{25} = 1$.

$z = 40$.

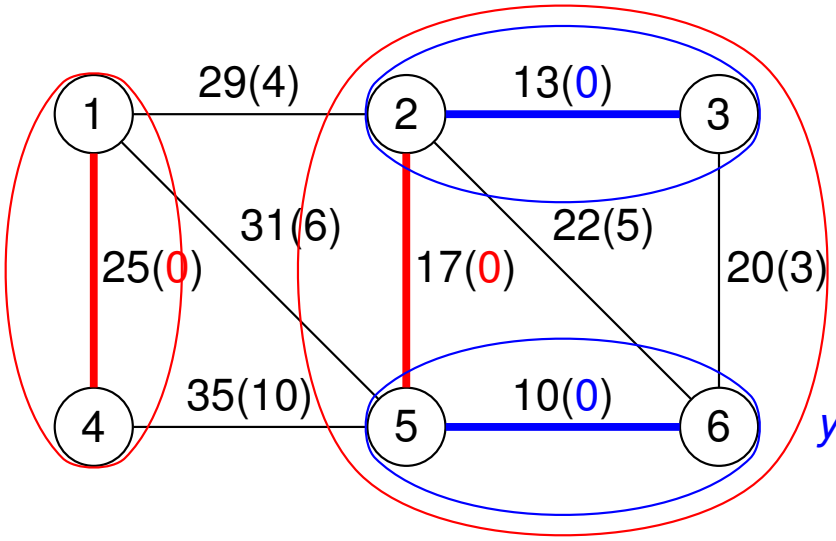
$y_V = 25$.

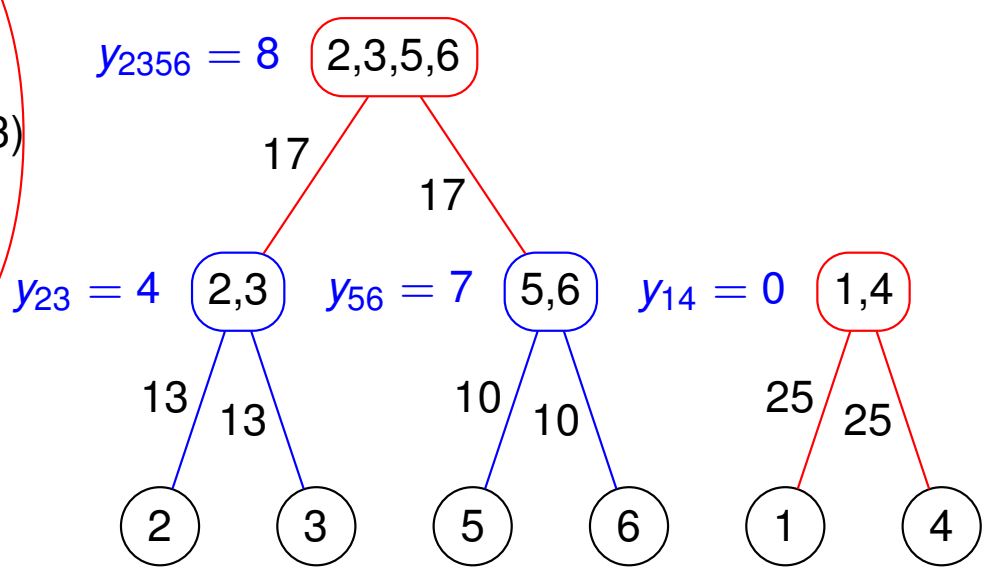
$w = -7 \times 1 - 4 \times 1 - 8 \times 3 +$
$+25 \times 5 = 90$.

Dual C.S.C.:

$y_V(k - (n-1)) = 25(3-5) = -50$.

An example: primal iteration 4





$k = 4$.

$x_{56} = x_{23} = x_{25} = x_{14} = 1$.

$z = 65$.

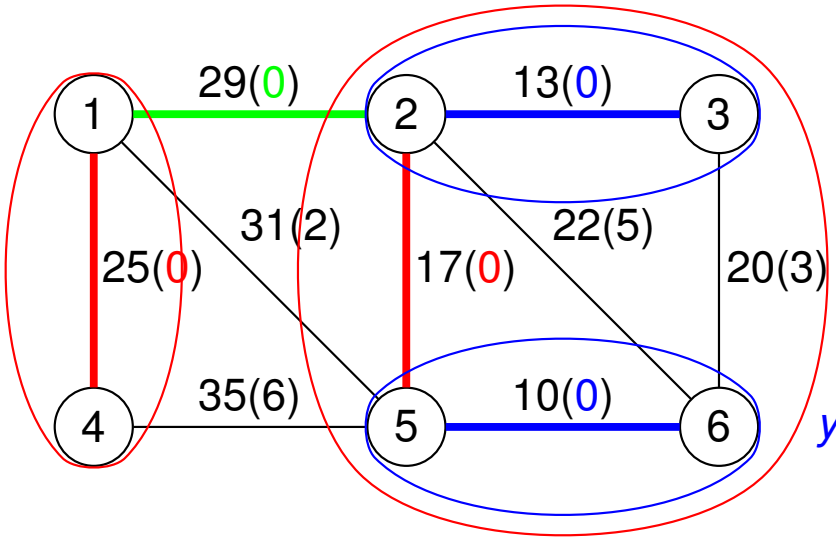
$y_V = 25$.

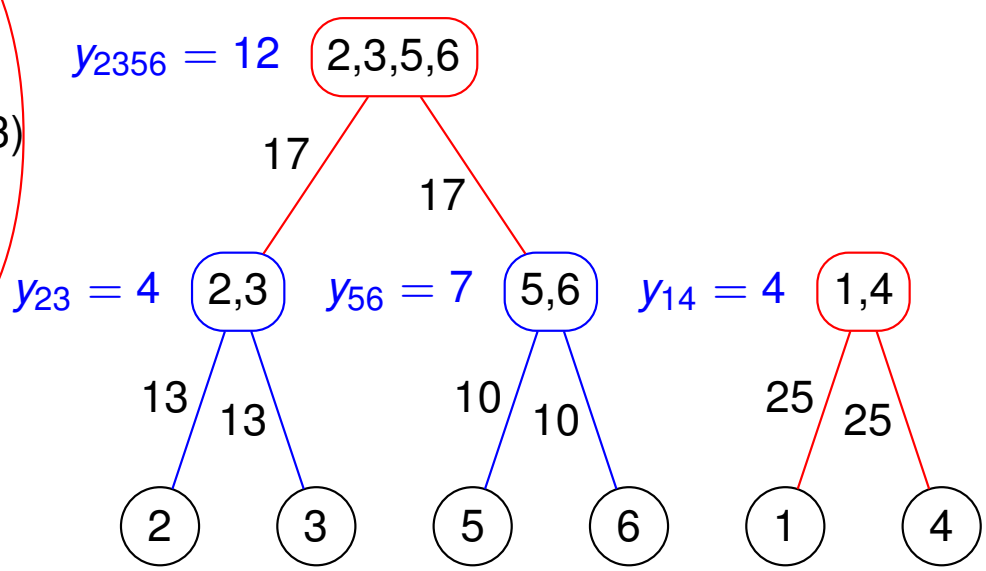
$w = -7 \times 1 - 4 \times 1 - 8 \times 3 +$
$-0 \times 1 + 25 \times 5 = 90$.

Dual C.S.C.:

$y_V(k - (n-1)) = 25(4-5) = -25$.

An example: dual iteration 5





$k = 4$.

$x_{56} = x_{23} = x_{25} = x_{14} = 1$.

$z = 65$.

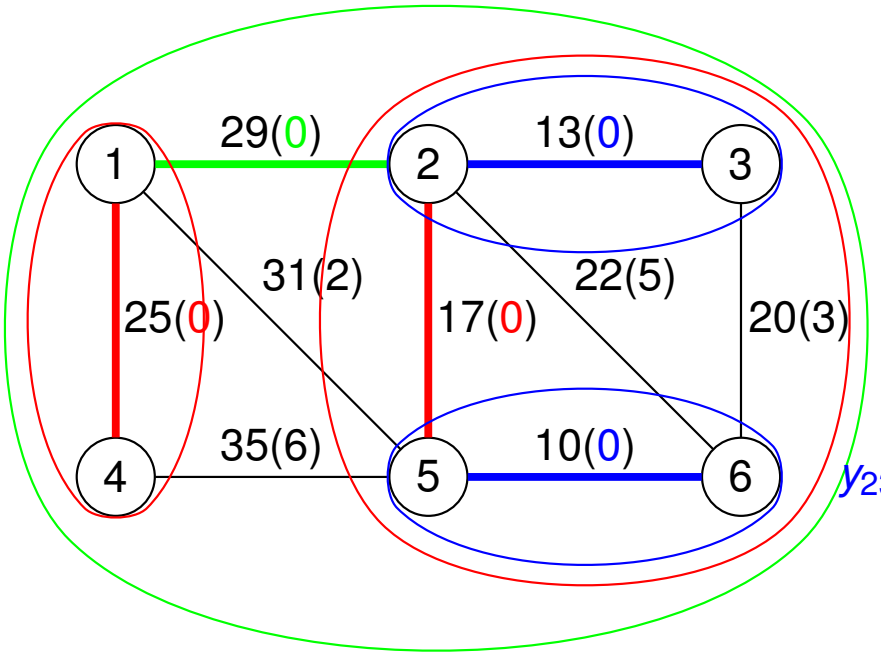
$y_V = 29$.

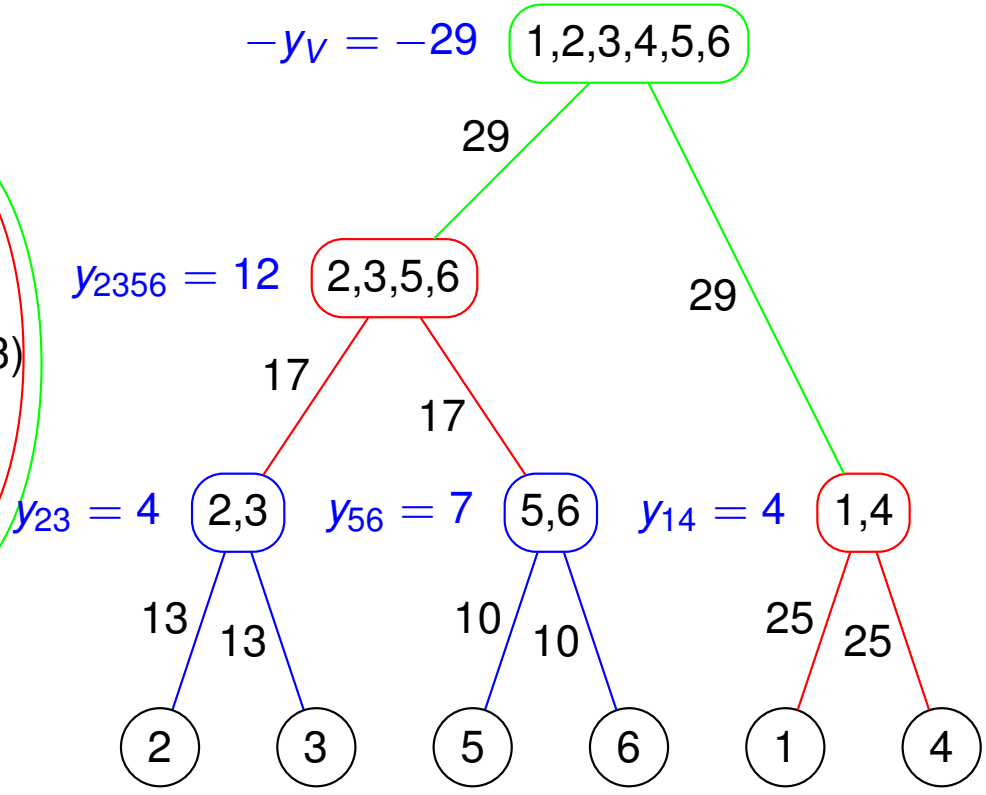
$w = -7 \times 1 - 4 \times 1 - 12 \times 3 +$
$-4 \times 1 + 29 \times 5 = 94$.

Dual C.S.C.:

$y_V(k - (n-1)) = 29(4-5) = -29$.

An example: primal iteration 5





$k = 5$ (*feasible*!).

$x_{23} = x_{56} = x_{25} = x_{14} = x_{12} = 1$.

$z = 94$.

$y_V = 29$.

$w = -7 \times 1 - 4 \times 1 - 12 \times 3 +$
$-4 \times 1 + 29 \times 5 = 94$.

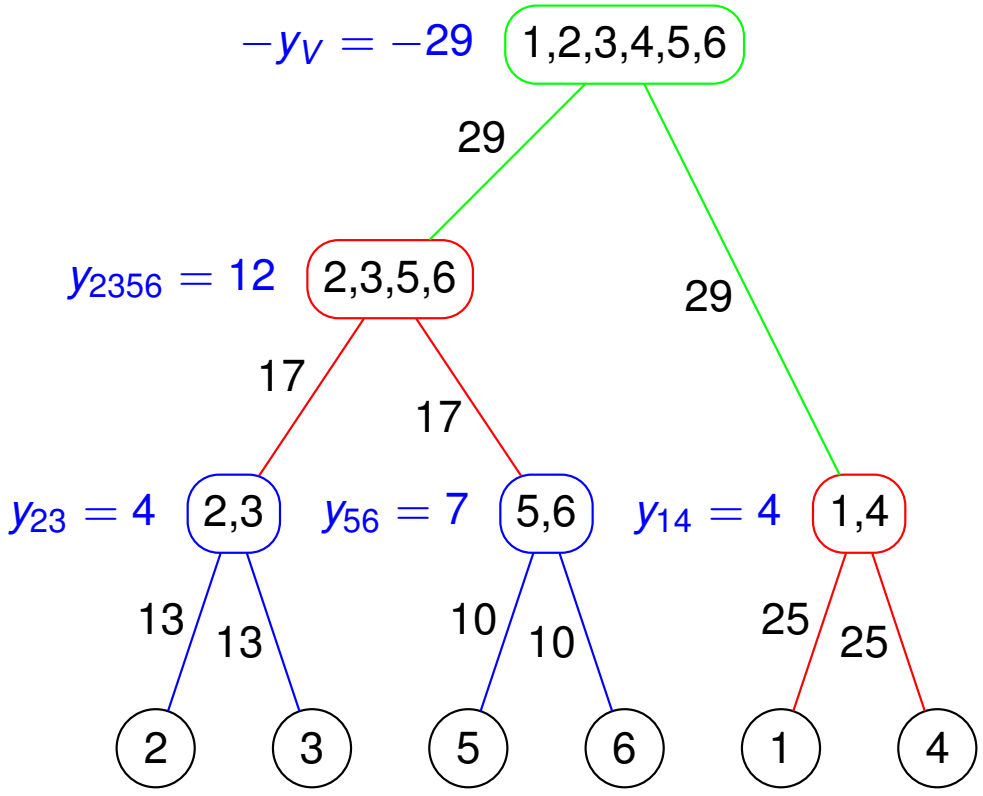
Dual C.S.C.:

$y_V(k - (n-1)) = 29(5-5) = 0$.

Dual optimal solution

Dual feasibility requires $\sum_{S \subset \mathcal{V}: e \in \mathcal{E}(S)} y_S + c_e \geq y_V \quad \forall e \in \mathcal{E}$.

For each connected component S, the value that the corresponding dual variable y_S takes is equal to the difference between the minimum edge cost in the cut $\delta(S)$ and the maximum edge cost in the minimum spanning tree of the induced subgraph $\mathcal{E}(S)$.



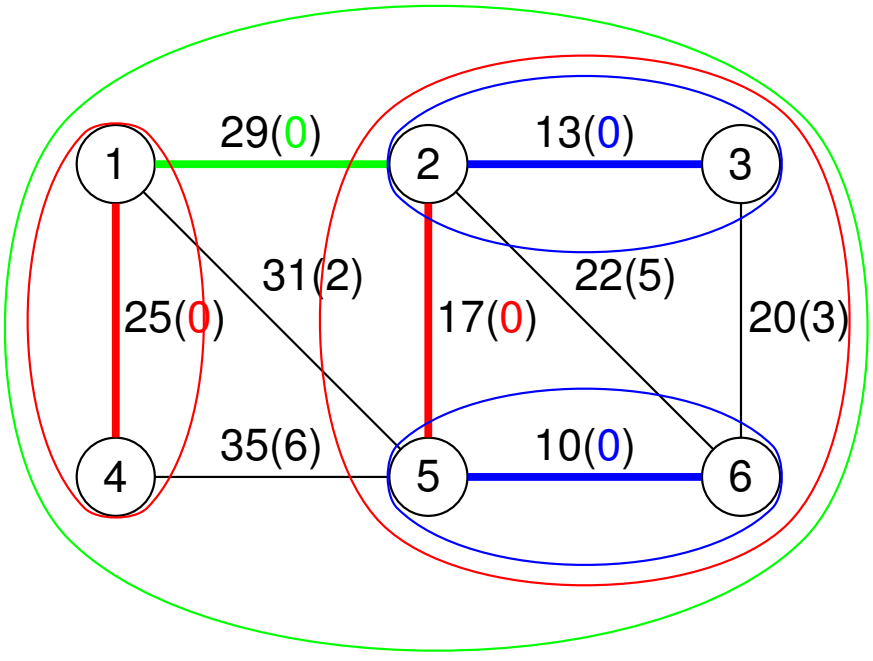
$y_V = 29$.

$w = -7 \times 1 - 4 \times 1 - 12 \times 3 + -4 \times 1 + 29 \times 5 = 94$.

Dual C.S.C.:

$y_V(k - (n-1)) = 29(5-5) = 0$.

Primal optimal solution



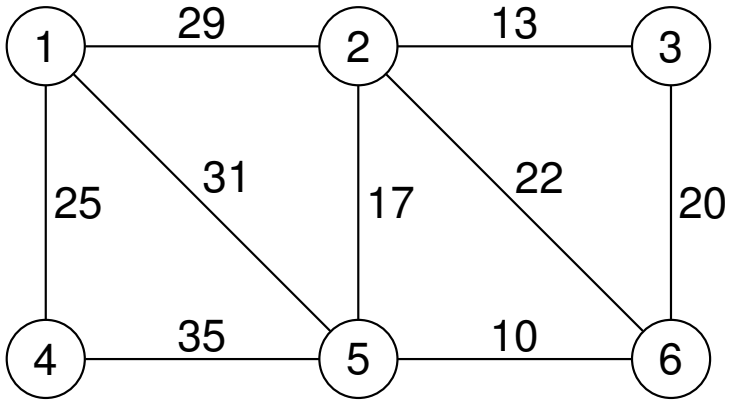
$k = 5$.
$x_{23} = x_{56} = x_{25} = x_{14} = x_{12} = 1$.
$z = 94$.

The reduced cost of each edge is the difference between its cost and the largest cost along the (unique) path between its endpoints in the spanning tree.

Edges with positive reduced cost do not belong to the minimum cost spanning tree, because there is a cycle in which they have the largest cost.

Owing to the dual ascent procedure (monotonic increase of y_V), edges are chosen in non-decreasing order according to their costs.

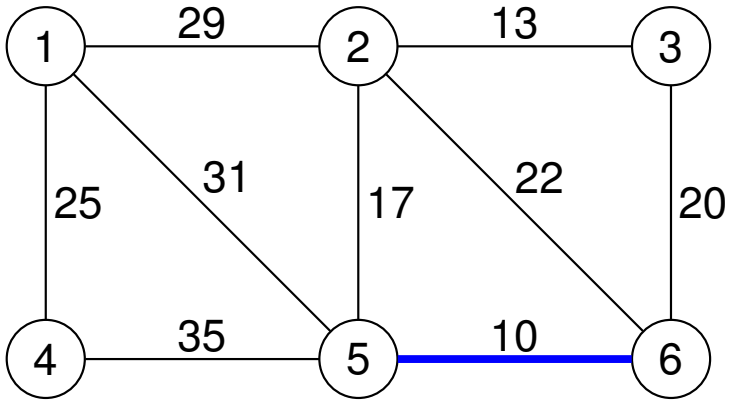
The algorithm and the data-structures



Edge	Cost	x
5,6	10	0
2,3	13	0
2,5	17	0
3,6	20	0
2,6	22	0
1,4	25	0
1,2	29	0
1,5	31	0
4,5	35	0
	z=0	k=0

Vertex	Head	Card.	List
1	1	1	1
2	2	1	2
3	3	1	3
4	4	1	4
5	5	1	5
6	6	1	6

Dual iteration 1

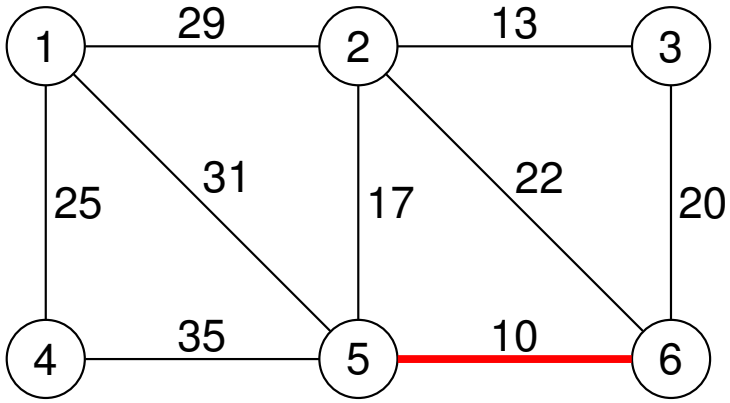


Edge	Cost	x
5,6	10	
2,3	13	
2,5	17	
3,6	20	
2,6	22	
1,4	25	
1,2	29	
1,5	31	
4,5	35	
	z=0	k=0

Vertex	Head	Card.	List
1	1	1	1
2	2	1	2
3	3	1	3
4	4	1	4
5	5	1	5
6	6	1	6

Edge [5, 6] accepted.
List 6 is appended to list 5.

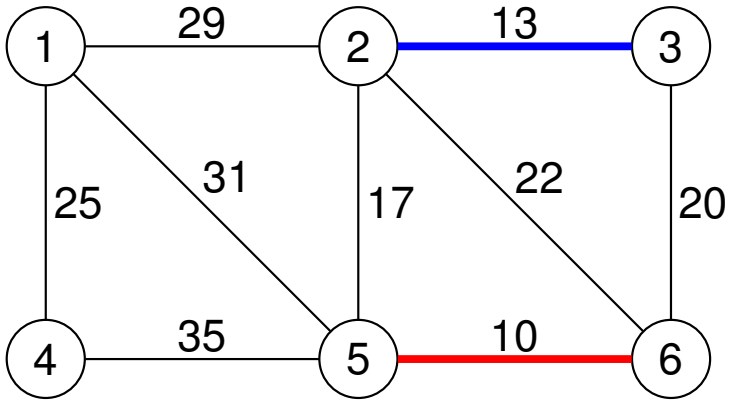
Primal iteration 1



Edge	Cost	x
5,6	10	1
2,3	13	
2,5	17	
3,6	20	
2,6	22	
1,4	25	
1,2	29	
1,5	31	
4,5	35	
	z=10	k=1

Vertex	Head	Card.	List
1	1	1	1
2	2	1	2
3	3	1	3
4	4	1	4
5	5	2	5 6
6	5	0	

Dual iteration 2

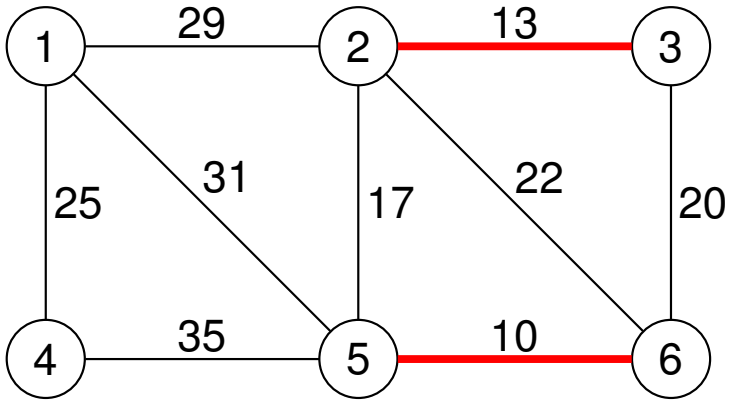


Edge	Cost	x
5,6	10	1
2,3	13	
2,5	17	
3,6	20	
2,6	22	
1,4	25	
1,2	29	
1,5	31	
4,5	35	
	z=10	k=1

Vertex	Head	Card.	List
1	1	1	1
2	2	1	2
3	3	1	3
4	4	1	4
5	5	2	5 6
6	5	0	

Edge [2, 3] is accepted.
List 3 is appended to list 2.

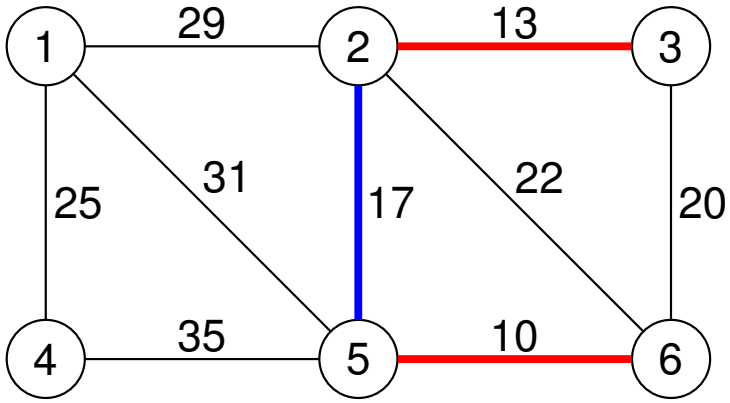
Primal iteration 2



Edge	Cost	x
5,6	10	1
2,3	13	1
2,5	17	
3,6	20	
2,6	22	
1,4	25	
1,2	29	
1,5	31	
4,5	35	
	z=23	k=2

Vertex	Head	Card.	List
1	1	1	1
2	2	2	2 3
3	2	0	
4	4	1	4
5	5	2	5 6
6	5	0	

Dual iteration 3

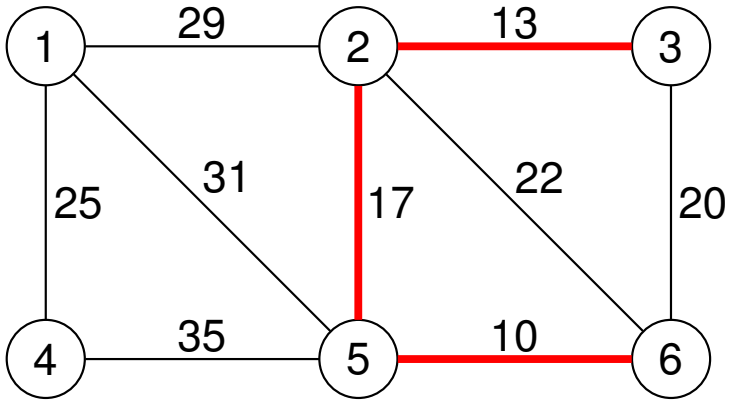


Edge	Cost	x
5,6	10	1
2,3	13	1
2,5	17	
3,6	20	
2,6	22	
1,4	25	
1,2	29	
1,5	31	
4,5	35	
	z=23	k=2

Vertex	Head	Card.	List
1	1	1	1
2	2	2	2 3
3	2	0	
4	4	1	4
5	5	2	5 6
6	5	0	

Edge [2, 5] is accepted.
List 5 is appended to list 2.

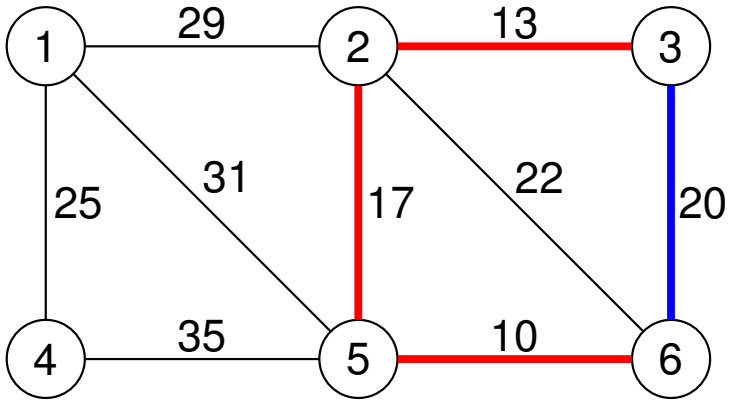
Primal iteration 3



Edge	Cost	x
5,6	10	1
2,3	13	1
2,5	17	1
3,6	20	
2,6	22	
1,4	25	
1,2	29	
1,5	31	
4,5	35	
	z=40	k=3

Vertex	Head	Card.	List
1	1	1	1
2	2	4	2 3 5 6
3	2	0	
4	4	1	4
5	2	0	
6	2	0	

Dual iteration 4 (part 1)

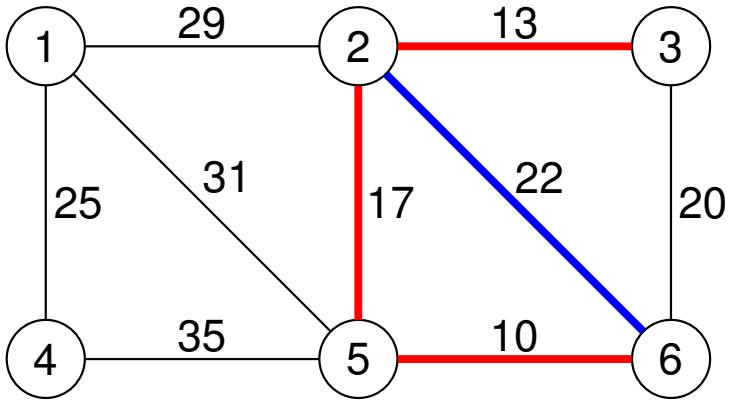


Edge	Cost	x
5,6	10	1
2,3	13	1
2,5	17	1
3,6	20	
2,6	22	
1,4	25	
1,2	29	
1,5	31	
4,5	35	
	z=40	k=3

Vertex	Head	Card.	List
1	1	1	1
2	2	4	2 3 5 6
3	2	0	
4	4	1	4
5	2	0	
6	2	0	

Edge [3, 6] is rejected.
Both its endpoints belong to list 2.

Dual iteration 4 (part 2)

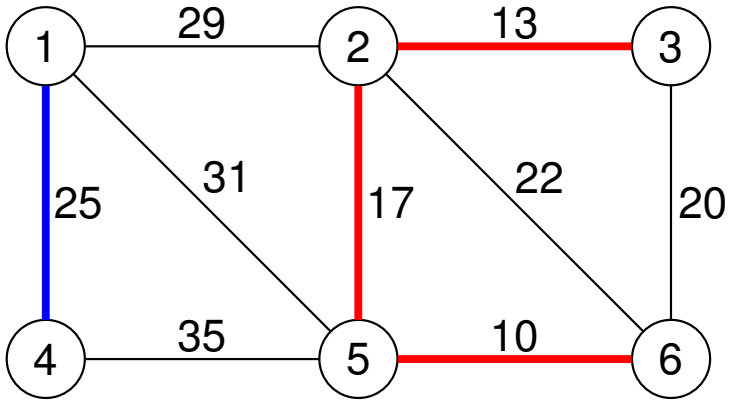


Edge	Cost	x
5,6	10	1
2,3	13	1
2,5	17	1
3,6	20	0
2,6	22	
1,4	25	
1,2	29	
1,5	31	
4,5	35	
	z=40	k=3

Vertex	Head	Card.	List
1	1	1	1
2	2	4	2 3 5 6
3	2	0	
4	4	1	4
5	2	0	
6	2	0	

Edge [2, 6] is rejected.
Both its endpoints belong to list 2.

Dual iteration 4 (part 3)

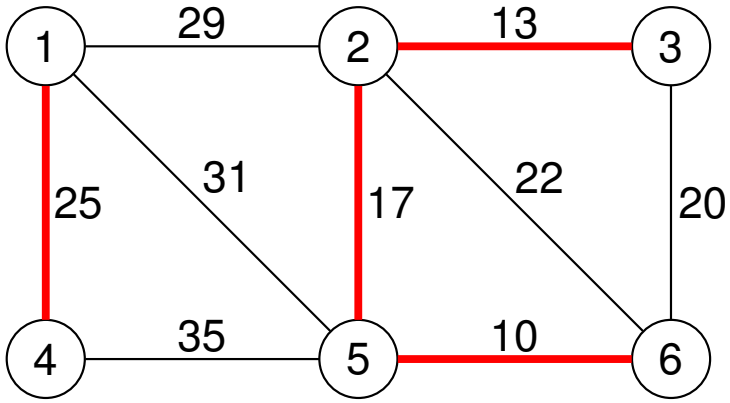


Edge	Cost	x
5,6	10	1
2,3	13	1
2,5	17	1
3,6	20	0
2,6	22	0
1,4	25	
1,2	29	
1,5	31	
4,5	35	
	z=40	k=3

Vertex	Head	Card.	List
1	1	1	1
2	2	4	2 3 5 6
3	2	0	
4	4	1	4
5	2	0	
6	2	0	

Edge [1, 4] is accepted.
List 4 is appended to list 1.

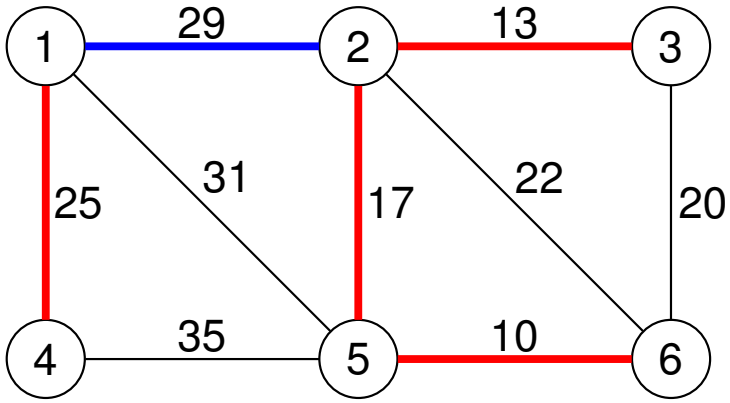
Primal iteration 4



Edge	Cost	x
5,6	10	1
2,3	13	1
2,5	17	1
3,6	20	0
2,6	22	0
1,4	25	1
1,2	29	
1,5	31	
4,5	35	
	z=65	k=4

Vertex	Head	Card.	List
1	1	2	1 4
2	2	4	2 3 5 6
3	2	0	
4	1	0	
5	2	0	
6	2	0	

Dual iteration 5

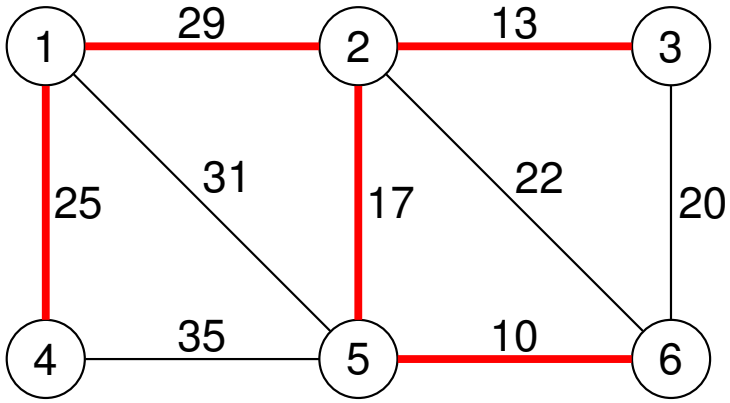


Edge	Cost	x
5,6	10	1
2,3	13	1
2,5	17	1
3,6	20	0
2,6	22	0
1,4	25	1
1,2	29	
1,5	31	
4,5	35	
	z=65	k=4

Vertex	Head	Card.	List
1	1	2	1 4
2	2	4	2 3 5 6
3	2	0	
4	1	0	
5	2	0	
6	2	0	

Edge [1, 2] is accepted.
List 1 is appended to list 2.

Primal iteration 5



Edge	Cost	x
5,6	10	1
2,3	13	1
2,5	17	1
3,6	20	0
2,6	22	0
1,4	25	1
1,2	29	1
1,5	31	
4,5	35	
	z^*=94	k=5

Vertex	Head	Card.	List
1	2	0	
2	2	6	2 3 5 6 1 4
3	2	0	
4	2	0	
5	2	0	
6	2	0	

The algorithm terminates: $k = 5$.
All nodes belong to the same list.