In [133]:

```
import matplotlib.pylab as plt
import matplotlib
import numpy as np
import math
import networkx as nx
%matplotlib inline
```

На вход подается файл в формате
-- n - количество вершин в графе
--
-- M - матрица смежности nxn. (симметричная, т.к. граф неориентированный. На главной диагонали нули, т.к. в графе нет петель)
-- В $M[i][j]$ указан вес ребра (i, j)
-- 0 не на главной диагонали означает нулевой вес ребра, т.к. граф полный
-- $a_{00}|a_{01}|\dots a_{0n}|$
-- $a_{10}|a_{11}|\dots a_{1n}|$
-- ...
-- $a_{n0}|a_{n1}|\dots a_{nn}|$
--
-- Номера выделенных вершин $V_0 = \{l|0 \leq l \leq n-1\}$
-- $v_1|v_2|v_3|$

In [239]:

```
def solve_test(test): #Обработка входных данных
    f = open(test, 'r')
    n = int(f.readline().split('\n')[0])
    adjacency_matrix = [[0 for i in range(n)] for j in range(n)]
    f.readline()
    for i in range(n):
        adjacency_matrix[i] = f.readline().split('\n')[0]
    for i in range(n):
        adjacency_matrix[i] = adjacency_matrix[i].split(" ")
    #print(adjacency_matrix)
    for i in range(n):
        for j in range(n):
            adjacency_matrix[i][j] = int(adjacency_matrix[i][j])
    f.readline()
    chosen_vertices = f.readline().split('\n')[0].split(" ")
    m = len(chosen_vertices)
    for i in range(m):
        chosen_vertices[i] = int(chosen_vertices[i])
    chosen_adjacency_matrix = [[0 for i in range(m)] for j in range(m)]
    for i in range(m):
        for j in range(m):
            chosen_adjacency_matrix[i][j] = \
            adjacency_matrix[chosen_vertices[i]][chosen_vertices[j]]
    return chosen_adjacency_matrix #Возвращает матрицу смежности для выделенных вершин
```

In [240]:

```
def find_mst(chosen_adjacency_matrix):
    #поиск минимального остовного дерева при помощи алгоритма Прима
    # (т.к. граф полный, алгоритм работает за O(n^2) времени и использует O(n) памяти)
    # На вход подается матрица смежности только для выделенных вершин
    m = len(chosen_adjacency_matrix[0])
    used = [0 for i in range(m)]
    min_edge = [math.inf for i in range(m)]
    selected_edge = [-1 for i in range(m)]
    k = 0
    min_edge[0] = 0

    for i in range(m):
        v = -1
        for j in range(m):
            if(used[j] == 0 and (v == -1 or min_edge[j] < min_edge[v])):
                v = j

        used[v] = 1
        if(selected_edge[v] != -1):
            k += chosen_adjacency_matrix[v][selected_edge[v]]
            print("Edge:", v, selected_edge[v], "Weight:", \
                  chosen_adjacency_matrix[v][selected_edge[v]])

        for to in range(m):
            if(chosen_adjacency_matrix[v][to] < min_edge[to]):
                min_edge[to] = chosen_adjacency_matrix[v][to]
                selected_edge[to] = v;
    print("Steiner Tree weight", k)
```

In [241]:

```
chosen_adjacency_matrix = solve_test("test3.txt")
find_mst(chosen_adjacency_matrix)
```

```
Edge: 2 0 Weight: 4
Edge: 4 2 Weight: 3
Edge: 3 4 Weight: 1
Edge: 1 3 Weight: 3
Steiner Tree weight 11
```

In [ ]:

```

```