Student Competition- Aufgabe 2

Constantin Lüth - Dante Bartoli

Gruppenname: Kruskal

30.06.2023

Inhaltsverzeichnis

1 Aufgabe

Ein Erdbeerbauer besitzt zwei Felder, auf denen er jeweils Erdbeeren anpflanzt. Wegen einer Durre sind schon sehr viele Pflanzen gestorben, weshalb er die restlichen Pflanzen nun bewässern möchte. Die Bewässerung soll so erfolgen, dass möglichst wenig Wasserschlauch verlegt werden muss. Die Schläuche können sich beliebig verzweigen und eine Wasserabgabe kann an jedem beliebigen Punkt des Schlauches erfolgen.

2 Aufgabenteil a

Das kleine Feld (Feld 1) misst 10 x 10. Leider haben hier nur 15 Pflanzen uberlebt. Ihre ¨ Positionen sind in Abbildung 1 dargestellt. Die Koordinaten finden Sie in der Excel-Datei "Erdbeerfelder.xlsx" im Datenblatt "Feld 1".

2.1 Aufgabenteil a1

Aufgabe: Wie verlaufen die Schläuche, wenn sich die Wasserpumpe genau bei Pflanze 1 befindet?

Idee 1

Wir verwenden einen »minimal spannenden Baum«, um alle Pflanzen mit minimalen Wasserschlauchmetern zu bewässern.

Imports

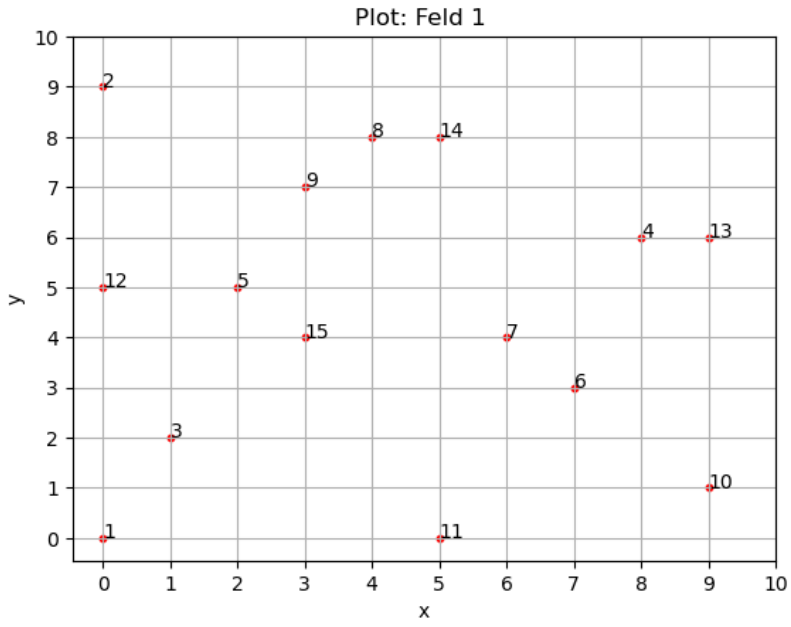
In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import networkx as nx
from scipy.spatial.distance import pdist, squareform
```

Koordinaten importieren

In [ ]:
```
df = pd.read_excel("Erdbeerfelder.xlsx", sheet_name=0, skiprows=1, usecols="C:D",names=["x","y"])
df = df.astype(float)
```

Erdbeer-Feld Plotten

In [ ]:
```
plt.scatter(df["x"], df["y"],s=10,c="red")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Plot: Feld 1 ")
plt.grid(True)
for i, txt in enumerate(df.index):
    plt.annotate(txt + 1, (df["x"][i], df["y"][i]))
plt.xticks(range(0, 11, 1))
plt.yticks(range(0, 11, 1))
plt.show()
```



Erstellung der Distanzmatrix (euklidische Distanz)

In [ ]:
```
# Berechnung der Distanzmatrix
dist_matrix = squareform(pdist(df))
```

Erstellen des Netzwerkes mit networkX

In [ ]:
```
# Erstellung des Netzwerks
G = nx.Graph()

# Hinzufügen der Knoten
for i in range(len(df)):
    G.add_node(i,x=df["x"][i], y=df["y"][i])
```

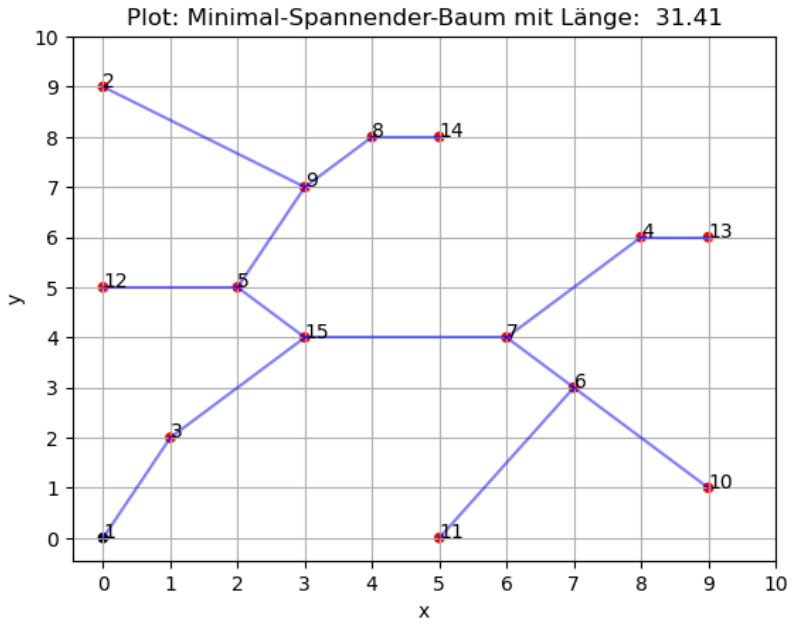
```
# Hinzufügen der Kanten mit den Distanzen als Gewicht
for i in range(len(df)):
    for j in range(len(df)):
        G.add_edge(i, j, weight=dist_matrix[i][j])
```

Berechnung des minimalen Spannbaums und des zugehörigen Schlauchlänge

In [ ]:
```
# Berechnung des minimalen Spannbaums
T = nx.minimum_spanning_tree(G)
total_length = 0
for u, v in T.edges():
    total_length += G[u][v]['weight']
```

Plotten des minimalen spannenden Baum

In [ ]:
```
colors = ["black" if (x, y) == (0, 0) else "red" for x, y in zip(df["x"], df["y"])]
plt.scatter(df["x"], df["y"], s=20, c=colors)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Plot: Minimal-Spannender-Baum mit Länge:  " + str(round(total_length, 2)))
plt.grid(True)

for i, txt in enumerate(df.index):
    plt.annotate(txt + 1, (df["x"][i], df["y"][i]))

for (u, v) in T.edges():
    x1, y1 = G.nodes[u]["x"], G.nodes[u]["y"]
    x2, y2 = G.nodes[v]["x"], G.nodes[v]["y"]
    plt.plot([x1, x2], [y1, y2], 'b-', alpha=0.5)

plt.xticks(range(0, 11, 1))
plt.yticks(range(0, 11, 1))

plt.show()
```



Ergebnis 1

Wir erhalten einen minimal spannenden Baum mit einer Länge von 31.41 Meter.

2.2 Aufgabenteil a2

Aufgabe: Was ändert sich an Ihrer Lösung, wenn sich die Wasserpumpe genau bei Pflanze 2 befindet?

Nichts, da es für den minimal-spannenden-Baum unerheblich ist, wo sich die Wasserpumpe befindet (vgl. Abbildung 2.1). Die Schläuche bleiben in der gleichen Position wie in Aufgabe a1.

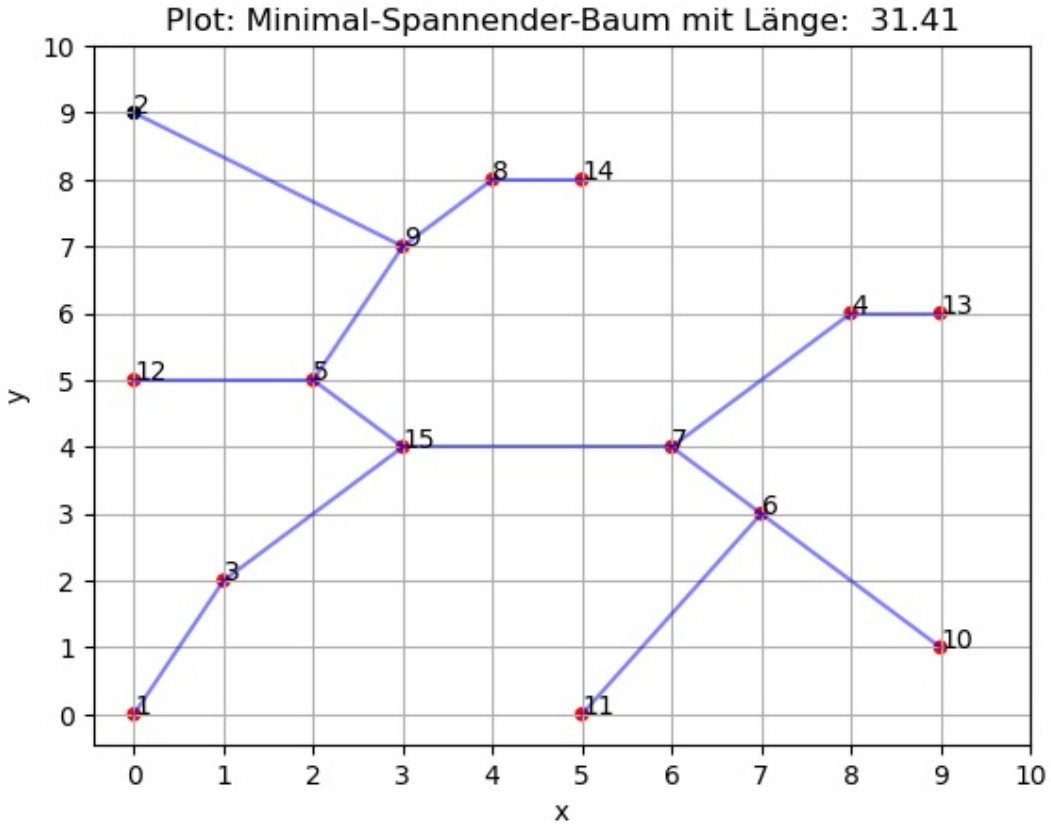


Abbildung 2.1: Minimal spannender Baum (Pumpe im Punkt 2)

2.3 Aufgabenteil a3

Aufgabe: Finden Sie eine möglichst gute Lösung fur den Fall, dass sich genau bei Pflanze 1 und in Koordinatenposition (10/10) eine Wasserpumpe befindet.

Imports

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import networkx as nx
from scipy.spatial.distance import pdist, squareform
```

Koordinaten importieren und hinzufügen der neuen Wasserpumpe ins DF

In [ ]:
```
df = pd.read_excel("Erdbeerfelder.xlsx", sheet_name=0, skiprows=1, usecols="C:D",names=["x","y"])
df = df.astype(float)
df= df.append({"x":10,"y":10},ignore_index=True)
```

```
/var/folders/4z/c2y3b6j9799gk1l74sc986mw0000gn/T/ipykernel_27561/3766332710.py:3: FutureWarning: The frame.append meth
od is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  df= df.append({"x":10,"y":10},ignore_index=True)
```

Erstellung der Distanzmatrix (euklidische Distanz)

In [ ]:
```
# Berechnung der Distanzmatrix
dist_matrix = squareform(pdist(df))
```

Erstellen des Netzwerkes mit networkX

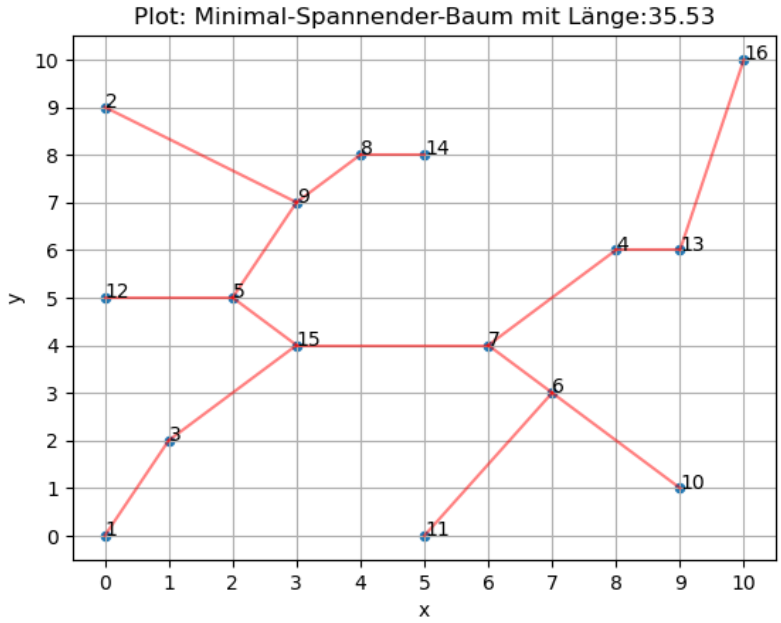
In [ ]:
```
# Erstellung des Netzwerks
G = nx.Graph()

# Hinzufügen der Knoten
for i in range(len(df)):
    G.add_node(i,x=df["x"][i], y=df["y"][i])

# Hinzufügen der Kanten mit den Distanzen als Gewicht
for i in range(len(df)):
    for j in range(len(df)):
        G.add_edge(i, j, weight=dist_matrix[i][j])
```

Berechnung des minimalen Spannbaums und des zugehörigen Schlauchlänge

In [ ]:
```
# Berechnung des minimalen Spannbaums
T = nx.minimum_spanning_tree(G)
total_length = 0
for u, v in T.edges():
    total_length += G[u][v]['weight']
```

Visualisierung

In [ ]:
```
# Scatter-Plot
plt.scatter(df["x"], df["y"], s=20)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Plot: Minimal-Spannender-Baum mit Länge:" + str(round(total_length,2)))
plt.grid(True)

# Anmerkungen für die Punkte
for i, txt in enumerate(df.index):
    plt.annotate(txt + 1, (df["x"][i], df["y"][i]))

# Zeichnen der Kanten des minimalen Spannbaums
for (u, v) in T.edges():
    x1, y1 = G.nodes[u]["x"], G.nodes[u]["y"]
    x2, y2 = G.nodes[v]["x"], G.nodes[v]["y"]
    plt.plot([x1, x2], [y1, y2], 'r-', alpha=0.5)

plt.xticks(range(0, 11, 1))
plt.yticks(range(0, 11, 1))

plt.show()
```



Hier sieht man den minimal spannenden Baum mit der Wasserpumpe in (0,0) und (10,0). Die Schlauchlänge beträgt 35,53 und stellt eine Obergrenze dar.

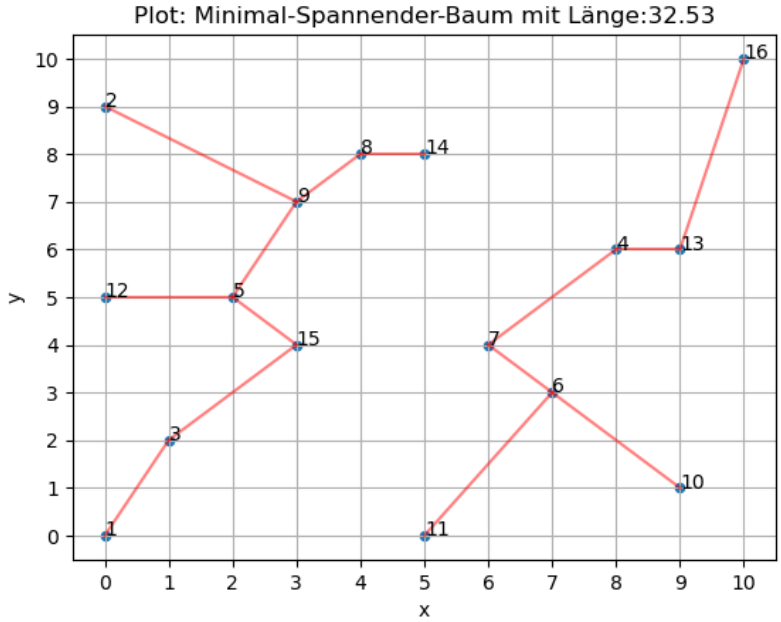
Idee 1

Es wird die Bedingung gestellt, dass beide Pumpen jeweils eine bestimmte Anzahl von Erdbeeren bewässern, d.h. es müssen zwei minimale spannende Bäume vorhanden sein. Es wird der Reihe nach durch die Knoten iteriert und geprüft, ob der Knoten zum (0,0)-Baum oder zum (10,10)-Baum gehört. Dazu wird getestet, welcher Baum nach Hinzufügen des Knotens weniger schwerer wurde ist.

In [ ]:
```
array__pumpe_1= []
array__pumpe_1.append(0)
array_pumpe_2= []
# Die Pumpe im Punkt (10,10) soll in einer eigenen Liste sein
```

```
array_pumpe_2.append(15)

for index in range(1, 15):
    temp_array_1 = array__pumpe_1 + [index]
    temp_array_2 = array_pumpe_2 + [index]

    #Checken welcher Baum minimaler nach einfügend des Knoten ist (< oder <= macht kein Unterschied aufs Endergebnis)
    if array_to_mst_value(temp_array_1) < array_to_mst_value(temp_array_2):
        array__pumpe_1.append(index)
    else:
        array_pumpe_2.append(index)
```

In [ ]:
```
plot_minimum_spanning_tree(array__pumpe_1,array_pumpe_2)
```



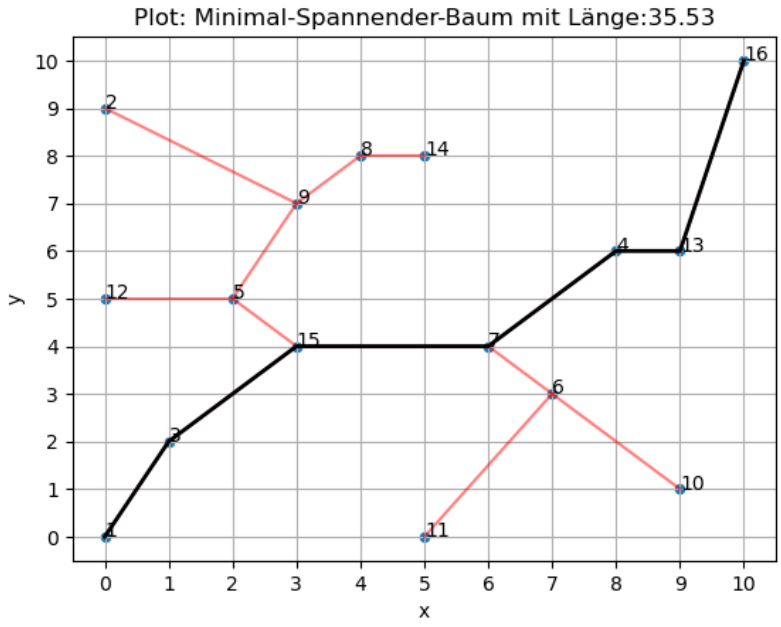
Ergebnis:
Mit Idee 1 erhalten wir zwei minimalspannde Bäume, welche zusammen eine schlauch Länge von 32,53 benötigen. Die Pumpen befinden sich in den Punkten 1 und 16.

Idee 2:
Einen minimalen spannenden Baum zwischen allen Punkten (auch Pumpe (10,10)) erstellen. Dann den Pfad zwischen (0,0) und (10,10) suchen. Iterativ jede Kante auf dem Pfad einmal entfernen. Dadurch entstehen zwei Bäume, welche in der Summe der Gewichte (Länge der Schläuche) minimal werden soll.

In [ ]:
```
# Finde den kürzesten Pfad zwischen den Punkten (0,0) und (10,10)
path = nx.shortest_path(T, source=0, target=15, weight='weight')
```

Visualisierung

In [ ]:
```
# Scatter-Plot
plt.scatter(df["x"], df["y"], s=20)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Plot: Minimal-Spannender-Baum mit Länge:" + str(round(total_length,2)))
plt.grid(True)

# Anmerkungen für die Punkte
for i, txt in enumerate(df.index):
    plt.annotate(txt + 1, (df["x"][i], df["y"][i]))

# Zeichnen der Kanten des minimalen Spannbaums
for (u, v) in T.edges():
    x1, y1 = G.nodes[u]["x"], G.nodes[u]["y"]
    x2, y2 = G.nodes[v]["x"], G.nodes[v]["y"]
    plt.plot([x1, x2], [y1, y2], 'r-', alpha=0.5)

# Path zeichnen
x_path = df.loc[path]["x"].tolist()
y_path = df.loc[path]["y"].tolist()
plt.plot(x_path, y_path, 'k-', linewidth=2)

plt.xticks(range(0, 11, 1))
plt.yticks(range(0, 11, 1))

plt.show()
```



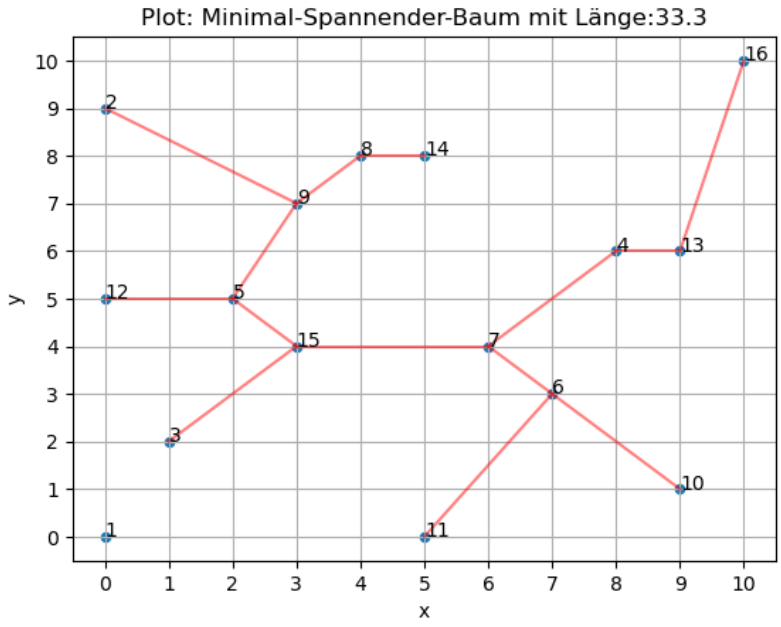
- Gehe jede Kante auf dem Weg iterativ ab
- Entferne sie
- Erhalte zwei neue minimal spannende Bäume
- Berechne die Summe der beiden längen

In [ ]:
```
min_path={}
# Iteriere durch alle Kanten auf dem Pfad
for i in range(len(path) - 1):
    dict_temp= {}
    T_copy = T.copy()

    # Entferne die aktuelle Kante i mit i+1
```

```
    T_copy.remove_edge(path[i], path[i+1])

    # Finde die zusammenhängenden Komponenten (d.h., die beiden Bäume)
    components = list(nx.connected_components(T_copy))

    # Der erste Baum ist die erste Komponente
    tree1 = T_copy.subgraph(components[0])

    # Der zweite Baum ist die zweite Komponente
    tree2 = T_copy.subgraph(components[1])

    # Alle Knoten des ersten Baums hinzufügen
    tree1_nodes = tree1.nodes()

    # Alle Knoten des zweiten Baums hinzufügen
    tree2_nodes = tree2.nodes()

    #Werte ins Dict hinzufügen
    dict_temp["tree1_nodes"]=list(tree1_nodes)
    dict_temp["tree2_nodes"]=list(tree2_nodes)
    dict_temp["value"]=array_to_mst_value(tree2_nodes)+array_to_mst_value(tree1_nodes)

    min_path[(path[i], path[i+1])]=dict_temp
```

In [ ]:
```
i=1
for edge, data in min_path.items():
    print("Abbildung "+ str(i) +" :")
    tree1_nodes = data['tree1_nodes']
    tree2_nodes = data['tree2_nodes']
    plot_minimum_spanning_tree(tree1_nodes,tree2_nodes)
    i+=1
```

```
Abbildung 1 :
```




```
Abbildung 2 :
```

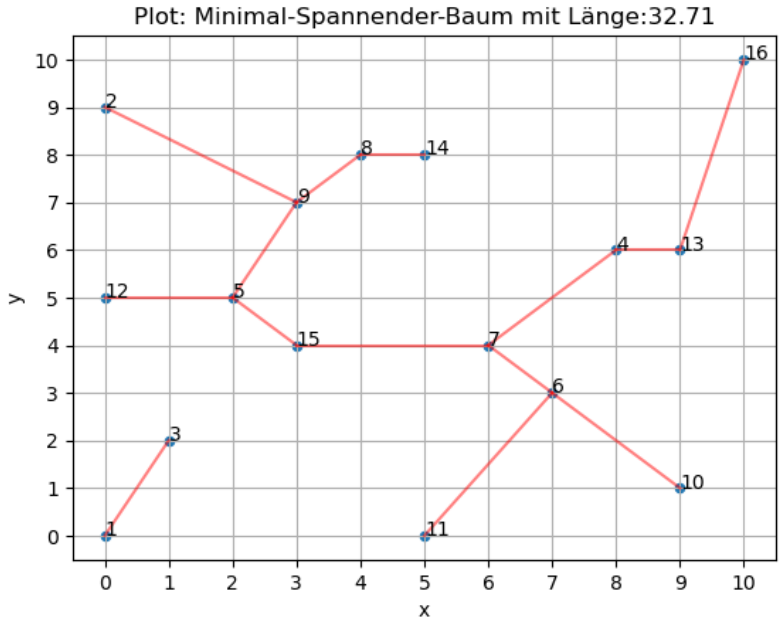


Abbildung 3 :

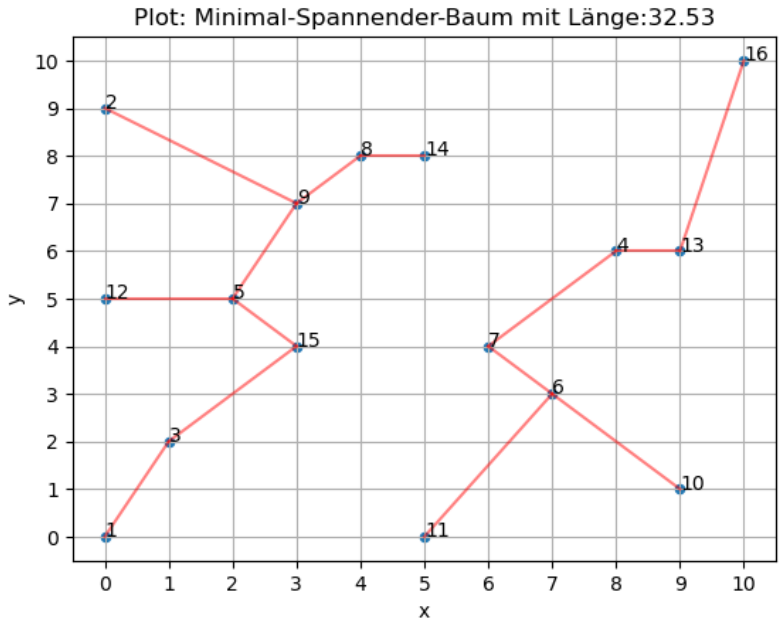


Abbildung 4 :

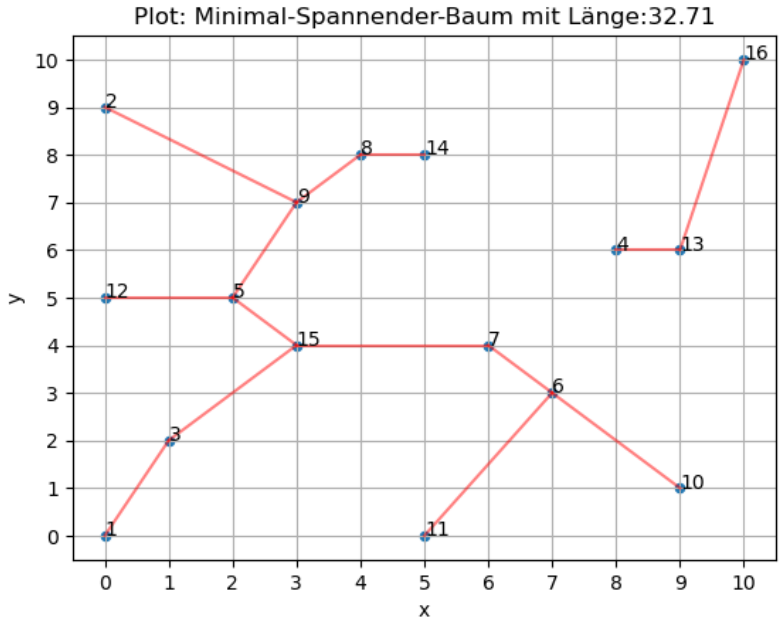


Abbildung 5 :

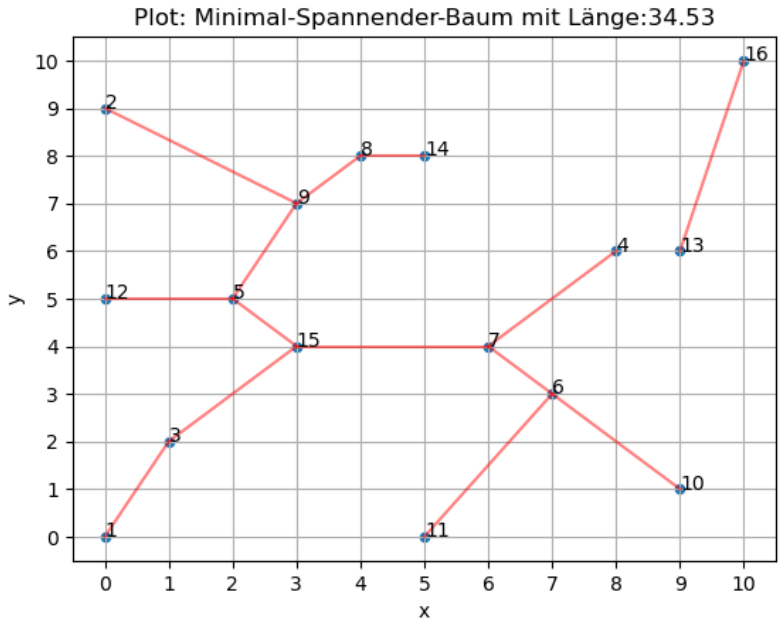
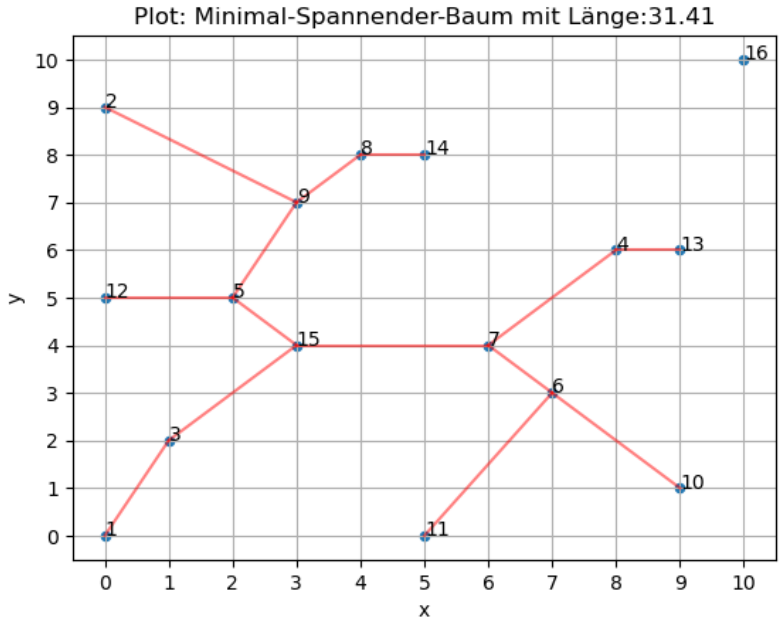


Abbildung 6 :



Ergebnis

Die kürzeste gesamt Schlauchlänge erhalten wir bei Abbildung 6.

Damit lässt sich Aufgabe a3) wie folgt beantworten:

- Die gesamt minimale Schlauchlänge beträgt 31.41
- Eine Wasserpumpe befindet sich im Punkt 1 und bewässert alle Punkte bis 15.
- die zweite Wasserpumpe befindet sich im Punkt 16, ist aber nicht angeschlossen

In [ ]:
```
def array_to_mst_value(array):
    # Erstellung des Netzwerks
    G_temp = nx.Graph()


    # Hinzufügen der Knoten
    for element in array:
        G_temp.add_node(element,x=df["x"][element], y=df["y"][element])

    # Hinzufügen der Kanten mit den Distanzen als Gewicht
    for i in array:
        for j in array:
            G_temp.add_edge(i, j, weight=dist_matrix[i][j])

    # Berechnung des minimalen Spannbaums
    T = nx.minimum_spanning_tree(G_temp)
    total_length_0 = 0
    for u, v in T.edges():
        total_length_0 += G_temp[u][v]['weight']

    return total_length_0
```

In [ ]:
```
def dataframe_to_Graph(df_x):
    # Berechnung der Distanzmatrix
    dist_matrix_x = squareform(pdist(df_x))

    # Erstellung des Netzwerks
    G_x = nx.Graph()

    # Hinzufügen der Knoten
    for i in range(len(df_x)):
        G_x.add_node(i,x=df_x["x"][i], y=df_x["y"][i])

    # Hinzufügen der Kanten mit den Distanzen als Gewicht
    for i in range(len(df_x)):
        for j in range(len(df_x)):
            G_x.add_edge(i, j, weight=dist_matrix_x[i][j])


    return G_x
```

In [ ]:
```
def plot_minimum_spanning_tree(array_pumpe_1, array_pumpe_2):
    # Erstes DataFrame mit den Indizes aus array_pumpe_1 erstellen
    df1 = df.loc[array_pumpe_1].copy().reset_index(drop=True)
    # Zweites DataFrame mit den Indizes aus array_pumpe_2 erstellen
    df2 = df.loc[array_pumpe_2].copy().reset_index(drop=True)

    Graph_Pumpe1 = dataframe_to_Graph(df1)
```

```
Graph_Pumpe2 = dataframe_to_Graph(df2)

# Berechnung des minimalen Spannbaums für Pumpe 1
T_1 = nx.minimum_spanning_tree(Graph_Pumpe1)
total_length_1 = sum(Graph_Pumpe1[u][v]['weight'] for u, v in T_1.edges())

# Berechnung des minimalen Spannbaums für Pumpe 2
T_2 = nx.minimum_spanning_tree(Graph_Pumpe2)
total_length_2 = sum(Graph_Pumpe2[u][v]['weight'] for u, v in T_2.edges())

# Scatter-Plot
plt.scatter(df["x"], df["y"], s=20)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Plot: Minimal-Spannender-Baum mit Länge:" + str(round(total_length_1 + total_length_2, 2)))
plt.grid(True)

# Punkte plotten. Index +1, da wir hier mit 0 anfangen zu zählen
for i, txt in enumerate(df.index):
    plt.annotate(txt + 1, (df["x"][i], df["y"][i]))

# Zeichnen der Kanten des minimalen Spannbaums für Pumpe 1
for (u, v) in T_1.edges():
    x1, y1 = Graph_Pumpe1.nodes[u]["x"], Graph_Pumpe1.nodes[u]["y"]
    x2, y2 = Graph_Pumpe1.nodes[v]["x"], Graph_Pumpe1.nodes[v]["y"]
    plt.plot([x1, x2], [y1, y2], 'r-', alpha=0.5)

# Zeichnen der Kanten des minimalen Spannbaums für Pumpe 2
for (u, v) in T_2.edges():
    x1, y1 = Graph_Pumpe2.nodes[u]["x"], Graph_Pumpe2.nodes[u]["y"]
    x2, y2 = Graph_Pumpe2.nodes[v]["x"], Graph_Pumpe2.nodes[v]["y"]
    plt.plot([x1, x2], [y1, y2], 'r-', alpha=0.5)

plt.xticks(range(0, 11, 1))
plt.yticks(range(0, 11, 1))
plt.show()
```

2.4 Aufgabenteil a4

Aufgabe: Finden Sie eine möglichst gute Lösung, wenn zwei Wasserpumpen benutzt werden sollen und sich diese an beliebiger Stelle befinden können. Beschreiben und begrunden Sie Ihr Vorgehen bei der Standortwahl fur die Wasserpumpen und bei der Entscheidungsfindung fur die Verlegung der Schläuche genau.

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import networkx as nx
from scipy.spatial.distance import pdist, squareform
from sklearn.cluster import KMeans
import numpy as np
from sklearn.neighbors import NearestNeighbors
```

Impotieren der Daten

In [ ]:
```
df = pd.read_excel("Erdbeerfelder.xlsx", sheet_name=0, skiprows=1, usecols="C:D",names=["x","y"])
df = df.astype(float)
```

Idee 1

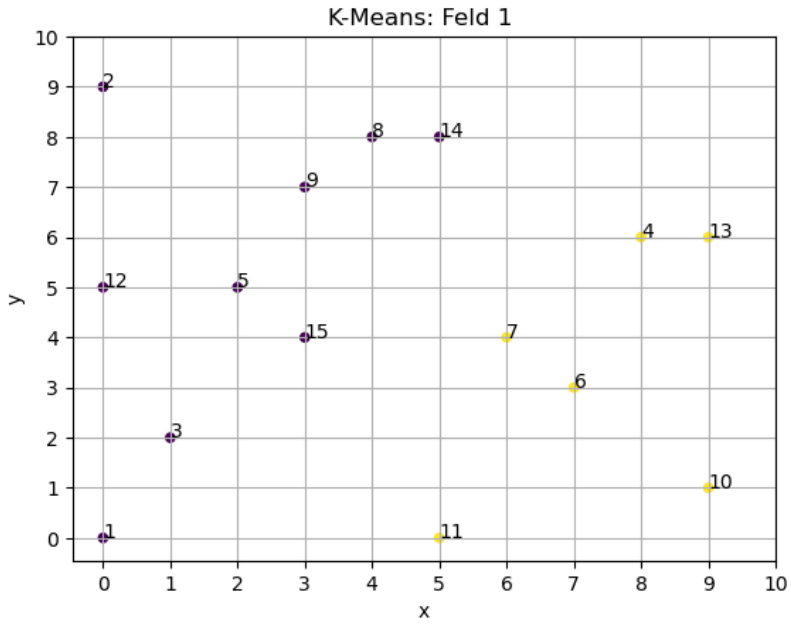
Die erste Idee, auf die wir kamen, war es, eine Clusteranalyse durchzuführen. Die Punkte mit k-means in zwei Cluster zuteilen und jedes Cluster mit einer Pumpe zu bewässern.

In [ ]:
```
# K-Means-Modell initialisieren und anpassen
kmeans = KMeans(n_clusters=2)
kmeans.fit(df)

# Cluster-Zuordnungen für jeden Datenpunkt erhalten
labels = kmeans.labels_

# Cluster-Zuordnungen zum DataFrame hinzufügen
df['Cluster'] = labels
```

Visualisierung

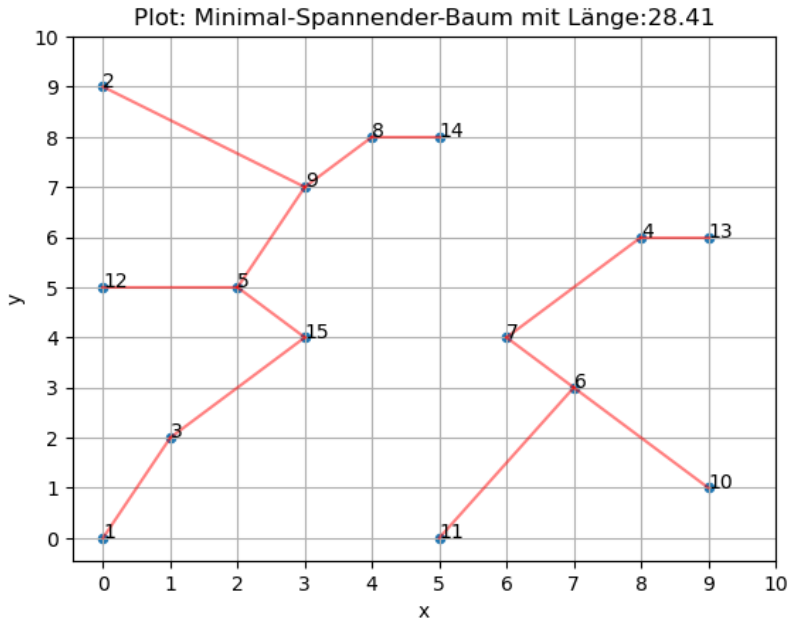
In [ ]:
```
plt.scatter(df["x"], df["y"],s=20,c=labels)
plt.xlabel("x")
plt.ylabel("y")
plt.title("K-Means: Feld 1 ")
plt.grid(True)
for i, txt in enumerate(df.index):
    plt.annotate(txt + 1, (df["x"][i], df["y"][i]))
plt.xticks(range(0, 11, 1))
```

```
plt.yticks(range(0, 11, 1))
plt.show()
```



In [ ]:
```
# Erzeuge zwei leere Arrays für die beiden Cluster
Cluster_1 = []
Cluster_2 = []

# Iteriere über den DataFrame und füge Indizes in die entsprechenden Arrays ein
for index, row in df.iterrows():
    cluster_label = row['Cluster']
    if cluster_label == 0:
        Cluster_1.append(index)
    elif cluster_label == 1:
        Cluster_2.append(index)
```

```
plot_minimum_spanning_tree(Cluster_1,Cluster_2)
```



Ergebnis

- Gesamtlänge von 28.41
- Eine Pumpe (egal wo) in {1,3,15,5,17,5,9,8,14,2} und eine in {11,6,10,7,4,13}

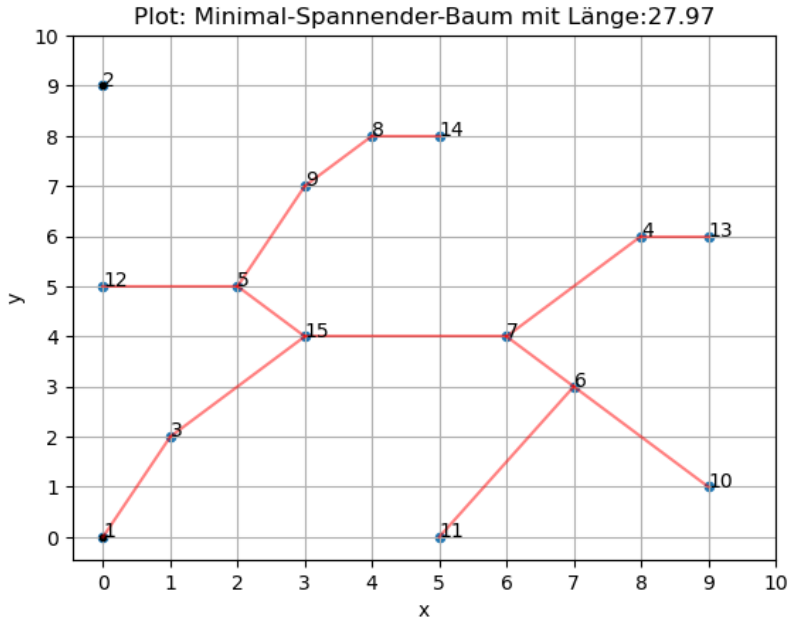
Idee2: NearestNeighbors

- Schaue welcher Punkt am weitesten von allen entfernt ist, also sein nächster Nachbar am weitesten.

In [ ]:
```
# NearestNeighbors-Modell erstellen (2, da jeder Punnkt sich selbst mit zählt, plus 1)
nbrs = NearestNeighbors(n_neighbors=2, metric='euclidean').fit(df)

# Abstände und Indizes des nächsten Nachbarn finden
distances, indices = nbrs.kneighbors(df)

# Index des Punktes mit dem größten Abstand zum nächsten Nachbarn finden
index_of_furthest_point = np.argmax(distances[:, 1])

# Den Index des am weitesten entfernten Punktes ausgeben
furthest_index = index_of_furthest_point
print("Der am weitesten entfernten Punktes ist:", furthest_index +1)

# Den am weitesten entfernten Punkt im DataFrame ausgeben
furthest_point = df.iloc[furthest_index]
print("Der am weitesten entfernte liegt in:", furthest_point)
```

```
Der am weitesten entfernten Punktes ist: 2
Der am weitesten entfernte liegt in: x          0.0
y          9.0
Cluster    0.0
Name: 1, dtype: float64
```

In [ ]:
```
# Array ohne furthest_index
array_without_furthest = df.drop(furthest_index).index

# Array nur mit furthest_index
furthest_index_array = np.array([furthest_index])


# Punkt des am weitesten entfernten Punktes plotten
plt.plot(furthest_point['x'], furthest_point['y'], 'ko',markersize=3)

# Irgendein Punkt schwarz im anderen Baum plotten, der die andere Pumpte darstellen soll
plt.plot(df["x"][array_without_furthest[0]], df["x"][array_without_furthest[0]], 'ko',markersize=3)

# Methode plot_minimum_spanning_tree aufrufen
plot_minimum_spanning_tree(array_without_furthest, furthest_index_array)

# Plot anzeigen
plt.show()
```



Ergebnis

- Gesamtlänge von 27.97
- Eine Pumpe (egal wo) in {1,3,15,5,17,5,9,8,14,11,6,10,7,4,13} und eine in {2}

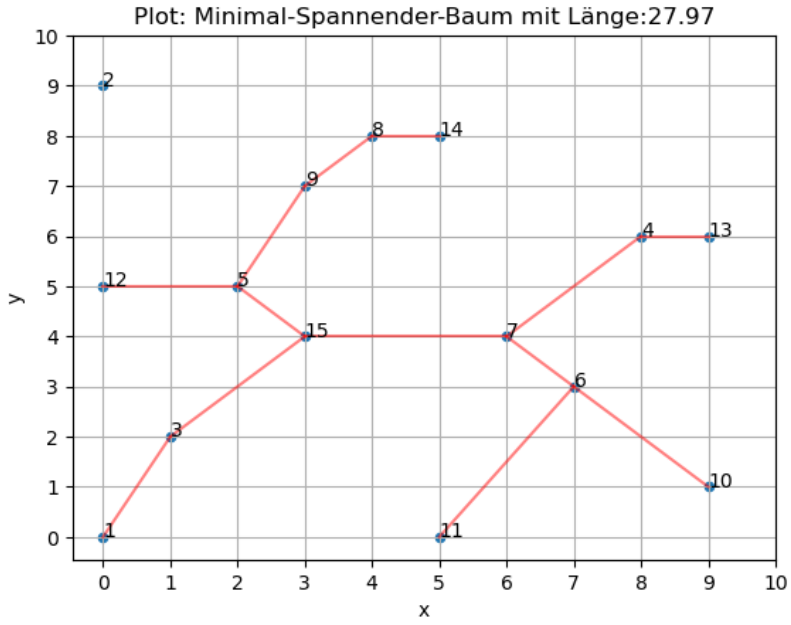
-> Ergebnis aber nicht für *alle* Fälle minimal.
-> Learning: Entferne immer die längste Kannte im MST um zwei neue Bäume daraus zuerhalten, die im Summe mnimal sind

Idee 3: Entferne längste Kannte im MST

In [ ]:
```
Graph=dataframe_to_Graph(df)

def student_competition_solver(Graph):
```

```
    # Berechnung des minimalen Spannbaums f
    T = nx.minimum_spanning_tree(Graph)

    # Suche die längste Kante
    longest_edge = max(T.edges(data=True), key=lambda x: x[2]['weight'])
    source_node, target_node, edge_attributes = longest_edge

    # Entferne die längste Kante
    T.remove_edge(source_node, target_node)

    # Teile den Graphen in zwei neue Bäume auf
    cut = list(nx.connected_components(T))

    # Knoten des ersten Baums
    MST1_nodes = list(cut[0])
    # Knoten des zweiten Baums
    MST2_nodes = list(cut[1])

    return MST1_nodes,MST2_nodes
```

In [ ]:
```
#Visualisierung

plot_minimum_spanning_tree(student_competition_solver(Graph)[0],student_competition_solver(Graph)[1])
```



Ergebnis

- Gesamtlänge von 27.97
- Eine Pumpe (egal wo) in {1,3,15,5,17,5,9,8,14,11,6,10,7,4,13} und eine in {2}
  -> ABER: Jetzt Dynamisch für alle Fälle.

In [ ]:
```python
def dataframe_to_Graph(df_x):
    # Berechnung der Distanzmatrix
    dist_matrix_x = squareform(pdist(df_x))

    # Erstellung des Netzwerks
    G_x = nx.Graph()
```

```python
    # Hinzufügen der Knoten
    for i in range(len(df_x)):
        G_x.add_node(i,x=df_x["x"][i], y=df_x["y"][i])

    # Hinzufügen der Kanten mit den Distanzen als Gewicht
    for i in range(len(df_x)):
        for j in range(len(df_x)):
            G_x.add_edge(i, j, weight=dist_matrix_x[i][j])


    return G_x
```

In [ ]:
```python
def plot_minimum_spanning_tree(array_pumpe_1, array_pumpe_2):
    # Erstes DataFrame mit den Indizes aus array_pumpe_1 erstellen
    df1 = df.loc[array_pumpe_1].copy().reset_index(drop=True)
    # Zweites DataFrame mit den Indizes aus array_pumpe_2 erstellen
    df2 = df.loc[array_pumpe_2].copy().reset_index(drop=True)

    Graph_Pumpe1 = dataframe_to_Graph(df1)
    Graph_Pumpe2 = dataframe_to_Graph(df2)

    # Berechnung des minimalen Spannbaums für Pumpe 1
    T_1 = nx.minimum_spanning_tree(Graph_Pumpe1)
    total_length_1 = sum(Graph_Pumpe1[u][v]['weight'] for u, v in T_1.edges())

    # Berechnung des minimalen Spannbaums für Pumpe 2
    T_2 = nx.minimum_spanning_tree(Graph_Pumpe2)
    total_length_2 = sum(Graph_Pumpe2[u][v]['weight'] for u, v in T_2.edges())

    # Scatter-Plot
    plt.scatter(df["x"], df["y"], s=20)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Plot: Minimal-Spannender-Baum mit Länge:" + str(round(total_length_1 + total_length_2, 2)))
    plt.grid(True)

    # Punkte plotten. Index +1, da wir hier mit 0 anfangen zu zählen
    for i, txt in enumerate(df.index):
        plt.annotate(txt + 1, (df["x"][i], df["y"][i]))

    # Zeichnen der Kanten des minimalen Spannbaums für Pumpe 1
    for (u, v) in T_1.edges():
        x1, y1 = Graph_Pumpe1.nodes[u]["x"], Graph_Pumpe1.nodes[u]["y"]
```

```
        x2, y2 = Graph_Pumpe1.nodes[v]["x"], Graph_Pumpe1.nodes[v]["y"]
        plt.plot([x1, x2], [y1, y2], 'r-', alpha=0.5)

    # Zeichnen der Kanten des minimalen Spannbaums für Pumpe 2
    for (u, v) in T_2.edges():
        x1, y1 = Graph_Pumpe2.nodes[u]["x"], Graph_Pumpe2.nodes[u]["y"]
        x2, y2 = Graph_Pumpe2.nodes[v]["x"], Graph_Pumpe2.nodes[v]["y"]
        plt.plot([x1, x2], [y1, y2], 'r-', alpha=0.5)

    plt.xticks(range(0, 11, 1))
    plt.yticks(range(0, 11, 1))
    plt.show()
```

3 Aufgabenteil b

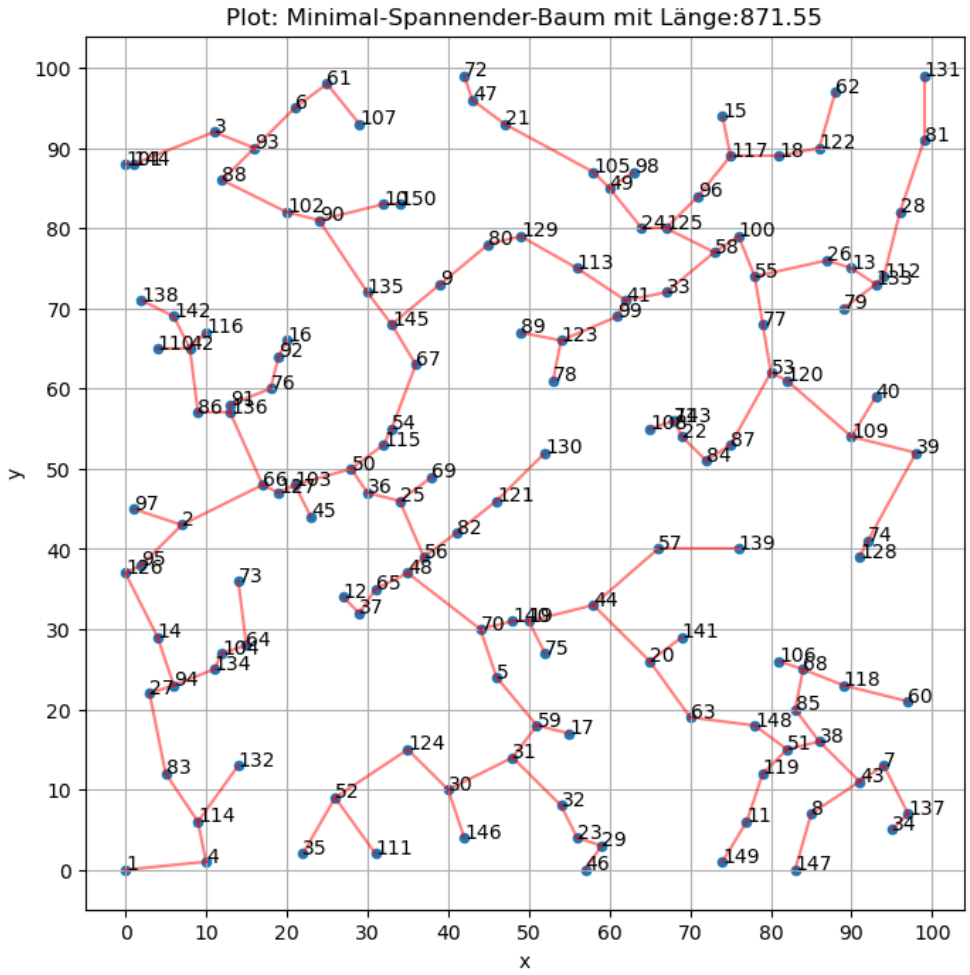
Das große Feld (Feld 2) misst 100 x 100. Hier haben 150 Pflanzen uberlebt. Ihre Positionen ¨ sind in Abbildung 2 gegeben, die Koordinaten finden Sie im Datenblatt ”Feld 2” der ExcelDatei ”Erdbeerfelder.xlsx”. Finden Sie auch hier eine möglichst gute Lösung, wenn zwei Wasserpumpen benutzt werden sollen und sich diese an beliebiger Stelle befinden können. Beschreiben und begrunden Sie Ihr Vorgehen bei der Standortwahl für die Wasserpumpen ¨ und bei der Entscheidungsfindung fur die Verlegung der Schläuche. Wie haben Sie Ihr Vorgehen im Vergleich zu a4) angepasst?

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import networkx as nx
from scipy.spatial.distance import pdist, squareform
from sklearn.cluster import KMeans
import numpy as np
from sklearn.neighbors import NearestNeighbors
```

Impotieren der Daten

In [ ]:
```
df = pd.read_excel("Erdbeerfelder.xlsx", sheet_name=1, skiprows=1, usecols="C:D",names=["x","y"])
df = df.astype(float)
```

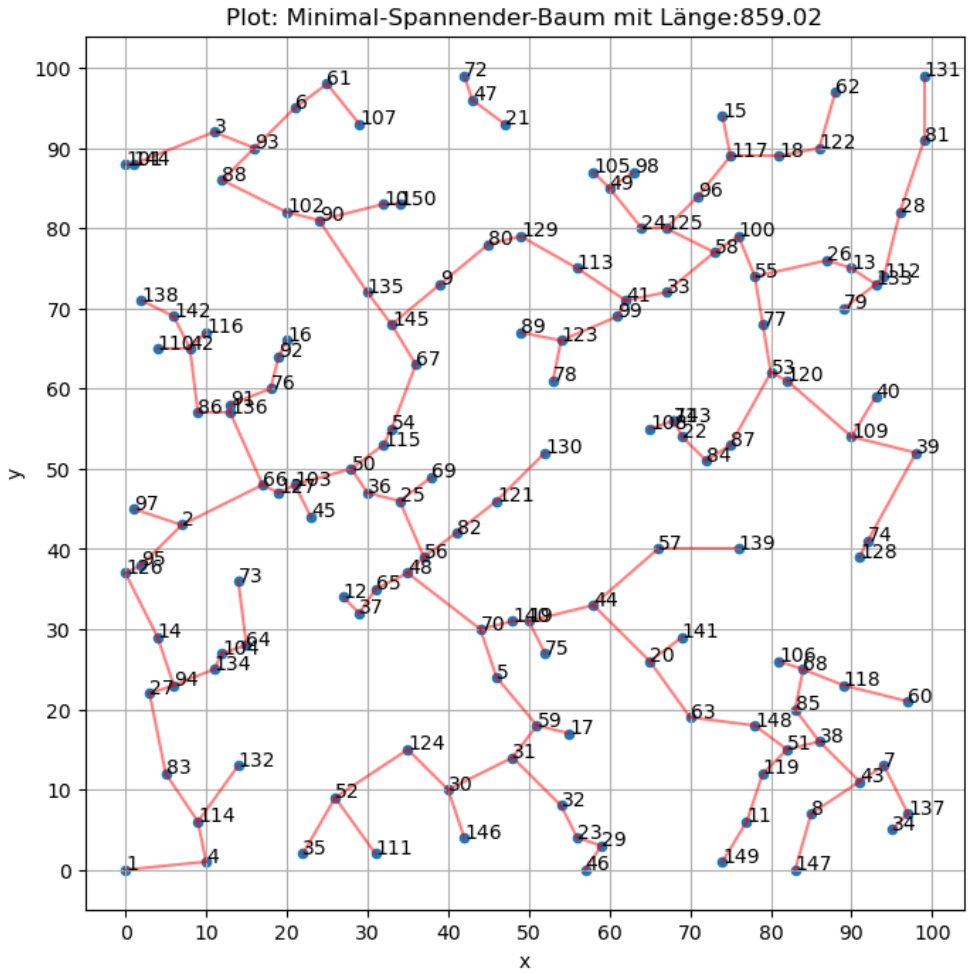
In [ ]:
```
plot_minimum_spanning_tree(np.array(df.index),[])
```



Idee: Entferne längste Kannte im MST

In [ ]:
```
Graph=dataframe_to_Graph(df)

def student_competition_solver(Graph):
    # Berechnung des minimalen Spannbaums f
    T = nx.minimum_spanning_tree(Graph)

    # Suche die längste Kante
    longest_edge = max(T.edges(data=True), key=lambda x: x[2]['weight'])
    source_node, target_node, edge_attributes = longest_edge

    # Entferne die längste Kante
    T.remove_edge(source_node, target_node)

    # Teile den Graphen in zwei neue Bäume auf
    cut = list(nx.connected_components(T))

    # Knoten des ersten Baums
    MST1_nodes = list(cut[0])
    # Knoten des zweiten Baums
    MST2_nodes = list(cut[1])

    return MST1_nodes,MST2_nodes
```

In [ ]:
```
#Visualisierung

plot_minimum_spanning_tree(student_competition_solver(Graph)[0],student_competition_solver(Graph)[1])
```



Ergebnis

- Gesamtlänge von 859.02, und damit sparen 2 Pumpen circa 12 (Einheit) Schlauch ein.
- Eine Pumpe (egal wo) in {21,47,21} und eine in in den resltichen Knoten

In [ ]:
```python
def dataframe_to_Graph(df_x):
    # Berechnung der Distanzmatrix
    dist_matrix_x = squareform(pdist(df_x))

    # Erstellung des Netzwerks
    G_x = nx.Graph()

    # Hinzufügen der Knoten
    for i in range(len(df_x)):
        G_x.add_node(i,x=df_x["x"][i], y=df_x["y"][i])

    # Hinzufügen der Kanten mit den Distanzen als Gewicht
    for i in range(len(df_x)):
        for j in range(len(df_x)):
            G_x.add_edge(i, j, weight=dist_matrix_x[i][j])


    return G_x
```

In [ ]:
```python
def plot_minimum_spanning_tree(array_pumpe_1, array_pumpe_2):
    # Erstes DataFrame mit den Indizes aus array_pumpe_1 erstellen
    df1 = df.loc[array_pumpe_1].copy().reset_index(drop=True)
    # Zweites DataFrame mit den Indizes aus array_pumpe_2 erstellen
    df2 = df.loc[array_pumpe_2].copy().reset_index(drop=True)

    Graph_Pumpe1 = dataframe_to_Graph(df1)
    Graph_Pumpe2 = dataframe_to_Graph(df2)

    # Berechnung des minimalen Spannbaums für Pumpe 1
    T_1 = nx.minimum_spanning_tree(Graph_Pumpe1)
    total_length_1 = sum(Graph_Pumpe1[u][v]['weight'] for u, v in T_1.edges())

    # Berechnung des minimalen Spannbaums für Pumpe 2
    T_2 = nx.minimum_spanning_tree(Graph_Pumpe2)
    total_length_2 = sum(Graph_Pumpe2[u][v]['weight'] for u, v in T_2.edges())

    # Scatter-Plot
```

```python
    plt.figure(figsize=(6, 6))
    plt.scatter(df["x"], df["y"], s=20)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Plot: Minimal-Spannender-Baum mit Länge:" + str(round(total_length_1 + total_length_2, 2)))
    plt.grid(True)

    # Punkte plotten. Index +1, da wir hier mit 0 anfangen zu zählen
    for i, txt in enumerate(df.index):
        plt.annotate(txt + 1, (df["x"][i], df["y"][i]))

    # Zeichnen der Kanten des minimalen Spannbaums für Pumpe 1
    for (u, v) in T_1.edges():
        x1, y1 = Graph_Pumpe1.nodes[u]["x"], Graph_Pumpe1.nodes[u]["y"]
        x2, y2 = Graph_Pumpe1.nodes[v]["x"], Graph_Pumpe1.nodes[v]["y"]
        plt.plot([x1, x2], [y1, y2], 'r-', alpha=0.5)

    # Zeichnen der Kanten des minimalen Spannbaums für Pumpe 2
    for (u, v) in T_2.edges():
        x1, y1 = Graph_Pumpe2.nodes[u]["x"], Graph_Pumpe2.nodes[u]["y"]
        x2, y2 = Graph_Pumpe2.nodes[v]["x"], Graph_Pumpe2.nodes[v]["y"]
        plt.plot([x1, x2], [y1, y2], 'r-', alpha=0.5)

    plt.xticks(range(0, 101, 10))
    plt.yticks(range(0, 101, 10))
    plt.show()
```

4 Bibliotheken

Bibliothek	**URL**
Pandas	`https://pandas.pydata.org/`
Matplotlib	`https://matplotlib.org/`
SciPy	`https://scipy.org/`
scikit-learn	`https://scikit-learn.org/stable/`
NumPy	`https://numpy.org/`
NetworkX	`https://networkx.org/`