00_slektstraer_og_grafer

October 12, 2025

[33]:
```
# ============================================================================
# GOOGLE COLAB SETUP / GOOGLE COLAB SETUP
# ============================================================================

# Sjekk om vi kjører i Google Colab
try:
    import google.colab
    IN_COLAB = True
    print("  Kjører i Google Colab - installerer avhengigheter...")
    print("  Running in Google Colab - installing dependencies...")

    # Installer nødvendige pakker
    import subprocess
    import sys
    try:
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-q",
                               "networkx", "matplotlib", "plotly", "pydantic",
                               "pyyaml", "pandas", "ipywidgets", "pillow",␣
 ↪"kaleido"])
        print("  Pakker installert")
    except Exception as e:
        print(f"  Pip install feilet: {e}")

    # Fjern eksisterende slektstre-mappe hvis den finnes
    import shutil
    import os
    if os.path.exists('/content/slektstre'):
        shutil.rmtree('/content/slektstre')
        print("  Fjernet eksisterende slektstre-mappe")

    # Klon repository
    try:
        subprocess.check_call(['git', 'clone', 'https://github.com/arvidl/
 ↪slektstre.git'])
        print("  Repository klonet")
    except Exception as e:
        print(f"  Git clone feilet: {e}")
```

```
# Legg til src-mappen til Python path og importer direkte
sys.path.insert(0, '/content/slektstre/src')
print("  Path lagt til")

# Importer slektstre-modulene direkte for å unngå navnekonflikt
import importlib.util
import types

# Først, fjern konfliktende moduler fra sys.modules
modules_to_remove = ['tree', 'models', 'localization']
for module_name in modules_to_remove:
    if module_name in sys.modules:
        del sys.modules[module_name]

# Last inn models.py først
try:
    spec = importlib.util.spec_from_file_location("slektstre_models", "/
↪content/slektstre/src/models.py")
    slektstre_models = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_models)

    # Opprett midlertidig models modul
    temp_models_module = types.ModuleType('models')
    temp_models_module.Person = slektstre_models.Person
    temp_models_module.Gender = slektstre_models.Gender
    temp_models_module.Ekteskap = slektstre_models.Ekteskap
    temp_models_module.FamilieData = slektstre_models.FamilieData
    sys.modules['models'] = temp_models_module

    print("  models.py lastet")
except Exception as e:
    print(f"  models.py feilet: {e}")

# Last inn localization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_localization",␣
↪"/content/slektstre/src/localization.py")
    slektstre_localization = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_localization)

    # Opprett midlertidig localization modul
    temp_localization_module = types.ModuleType('localization')
    temp_localization_module.t = slektstre_localization.t
    sys.modules['localization'] = temp_localization_module

    print("  localization.py lastet")
```

```
    except Exception as e:
        print(f"  localization.py feilet: {e}")

    # Last inn tree.py som slektstre_tree
    try:
        spec = importlib.util.spec_from_file_location("slektstre_tree", "/
↪content/slektstre/src/tree.py")
        slektstre_tree = importlib.util.module_from_spec(spec)
        spec.loader.exec_module(slektstre_tree)

        # Opprett midlertidig tree modul
        temp_tree_module = types.ModuleType('tree')
        temp_tree_module.Slektstre = slektstre_tree.Slektstre
        sys.modules['tree'] = temp_tree_module

        print("  tree.py lastet")
    except Exception as e:
        print(f"  tree.py feilet: {e}")

    # Last inn family_io.py
    try:
        spec = importlib.util.spec_from_file_location("slektstre_io", "/content/
↪slektstre/src/family_io.py")
        slektstre_io = importlib.util.module_from_spec(spec)
        spec.loader.exec_module(slektstre_io)
        print("  family_io.py lastet")
    except Exception as e:
        print(f"  family_io.py feilet: {e}")

    # Last inn visualization.py
    try:
        spec = importlib.util.spec_from_file_location("slektstre_viz", "/
↪content/slektstre/src/visualization.py")
        slektstre_viz = importlib.util.module_from_spec(spec)
        spec.loader.exec_module(slektstre_viz)
        print("  visualization.py lastet")
    except Exception as e:
        print(f"  visualization.py feilet: {e}")

    print("  Slektstre-moduler lastet inn i Colab")

except ImportError:
    IN_COLAB = False
    print("  Kjører lokalt / Running locally")
    import sys
    sys.path.append('../src')
except Exception as e:
```

```
    print(f"  Colab setup feilet: {e}")
    IN_COLAB = False
    print("  Fallback til lokal modus / Fallback to local mode")
    import sys
    sys.path.append('../src')

print(f"  Miljø: {'Google Colab' if IN_COLAB else 'Lokal'}")
print(f"  Environment: {'Google Colab' if IN_COLAB else 'Local'}")
```

[34]:
```
# Spill podcast direkte i notebook (uten å embedde data)
import IPython.display as ipd
import os
import gc

# Sjekk om IN_COLAB er definert, hvis ikke, sett til False
try:
    IN_COLAB
except NameError:
    IN_COLAB = False

# Prøv flere mulige paths for podcast-filen
possible_paths = []
if IN_COLAB:
    possible_paths = [
        "/content/slektstre/podcast/
 ↪Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3",
        "/content/slektstre/podcast/
 ↪Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.m4a"
    ]
else:
    possible_paths = [
        "podcast/
 ↪Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3",
        "podcast/
 ↪Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.m4a",
        "../podcast/
 ↪Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3",
        "../podcast/
 ↪Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.m4a",
        "./podcast/
 ↪Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3",
        "./podcast/
 ↪Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.m4a"
    ]

# Finn første eksisterende fil
```

```
podcast_path = None
for path in possible_paths:
    if os.path.exists(path):
        podcast_path = path
        break

if podcast_path:
    print("  Spiller podcast: Slektstre med Python og Grafteori")
    print("  Playing podcast: Family Trees with Python and Graph Theory")
    print(f"  Fil: {podcast_path}")

    # Initialiser audio_widget variabel
    audio_widget = None

    # Prøv IPython.display.Audio først (uten embed=False for lokale filer)
    try:
        audio_widget = ipd.Audio(podcast_path)
        display(audio_widget)
        print("  Audio-spiller opprettet")
    except Exception as e:
        print(f"  Audio-spiller feilet: {e}")
        print("  Prøv å åpne filen manuelt i nettleseren")

else:
    print("  Podcast-fil ikke funnet. Last ned fra GitHub repository.")
    print("  Podcast file not found. Download from GitHub repository.")
    print(f"  Søkte etter følgende paths:")
    for path in possible_paths:
        print(f"    - {path}")
    print(f"  Current directory: {os.getcwd()}")
```

[35]:
```
#  AUTOMATISK CLEANUP - Kjør denne cellen før commit/push (kun lokalt)
#  AUTOMATIC CLEANUP - Run this cell before commit/push (local only)

import os
import subprocess
import sys
import json

# Sjekk om vi kjører i Colab
try:
    IN_COLAB
except NameError:
    IN_COLAB = False

if IN_COLAB:
    print("  Cleanup-cellen hoppes over i Google Colab")
```

```
    print("  Cleanup cell skipped in Google Colab")
    print("  I Colab trenger du ikke å rydde opp før commit")
    print("  In Colab you don't need to cleanup before commit")
    print("  Colab-sessioner er midlertidige og påvirker ikke repository")
    print("  Colab sessions are temporary and don't affect repository")
else:
    def enhanced_cleanup_notebook():
        """Rydd opp i notebook-outputs med spesialhåndtering for embedded audio/
 ↪bilder"""

        # Prøv flere mulige paths for notebook
        possible_paths = [
            "notebooks/00_slektstraer_og_grafer.ipynb",
            "../notebooks/00_slektstraer_og_grafer.ipynb",
            "./notebooks/00_slektstre_og_grafer.ipynb",
            "00_slektstraer_og_grafer.ipynb"
        ]

        notebook_path = None
        for path in possible_paths:
            if os.path.exists(path):
                notebook_path = path
                break

        if not notebook_path:
            print(f"  Notebook ikke funnet. Prøvde følgende paths:")
            for path in possible_paths:
                print(f"    - {path}")
            print(f"  Current directory: {os.getcwd()}")
            return False

        print("  Starter automatisk cleanup av notebook...")
        print("  Starting automatic notebook cleanup...")
        print(f"  Bruker notebook: {notebook_path}")
        print(f"  Using notebook: {notebook_path}")

        # Få størrelse før cleanup
        size_before = os.path.getsize(notebook_path)
        print(f"  Størrelse før cleanup: {size_before / (1024*1024):.1f} MB")
        print(f"  Size before cleanup: {size_before / (1024*1024):.1f} MB")

        try:
            # Last inn notebook og fjern outputs manuelt
            with open(notebook_path, 'r', encoding='utf-8') as f:
                notebook = json.load(f)

            cells_modified = 0
```

```
            audio_removed = 0
            images_removed = 0

            # Prosesser hver celle
            for cell in notebook.get('cells', []):
                if cell.get('cell_type') == 'code' and 'outputs' in cell:
                    original_outputs = cell['outputs']
                    cell['outputs'] = []

                    # Sjekk om vi fjernet noe betydningsfullt
                    if original_outputs:
                        cells_modified += 1

                        # Tell hva vi fjerner
                        for output in original_outputs:
                            if 'data' in output:
                                data = output['data']

                                # Sjekk for embedded audio
                                for key in data.keys():
                                    if isinstance(data[key], list):
                                        for item in data[key]:
                                            if isinstance(item, str) and 'data:
↪audio' in item:
                                                audio_removed += 1
                                                break

                                # Sjekk for embedded bilder
                                for key in data.keys():
                                    if isinstance(data[key], list):
                                        for item in data[key]:
                                            if isinstance(item, str) and 'data:
↪image' in item:
                                                images_removed += 1
                                                break

            # Lagre den ryddede notebooken
            with open(notebook_path, 'w', encoding='utf-8') as f:
                json.dump(notebook, f, ensure_ascii=False, separators=(',', ':
↪'))

            # Få størrelse etter cleanup
            size_after = os.path.getsize(notebook_path)
            reduction = size_before - size_after

            print(f"  Enhanced cleanup fullført!")
            print(f"  Enhanced cleanup completed!")
```

```
            print(f"  Celler modifisert: {cells_modified}")
            print(f"  Cells modified: {cells_modified}")
            print(f"  Audio embeddings fjernet: {audio_removed}")
            print(f"  Audio embeddings removed: {audio_removed}")
            print(f"  Bilde embeddings fjernet: {images_removed}")
            print(f"  Image embeddings removed: {images_removed}")
            print(f"  Størrelse etter cleanup: {size_after / (1024*1024):.1f}␣
↪MB")
            print(f"  Size after cleanup: {size_after / (1024*1024):.1f} MB")
            print(f"  Reduksjon: {reduction / (1024*1024):.1f} MB")
            print(f"  Reduction: {reduction / (1024*1024):.1f} MB")

            if reduction > 1024 * 1024:  # Mer enn 1MB reduksjon
                print("  Betydelig størrelsesreduksjon oppnådd!")
                print("  Significant size reduction achieved!")
            elif reduction > 0:
                print("  Noe cleanup utført")
                print("  Some cleanup performed")
            else:
                print("  Ingen betydelig cleanup nødvendig")
                print("  No significant cleanup needed")

            print("  Notebook er nå klar for commit/push!")
            print("  Notebook is now ready for commit/push!")

            return True

        except Exception as e:
            print(f"  Cleanup feilet: {e}")
            return False

    # Kjør enhanced cleanup automatisk
    cleanup_success = enhanced_cleanup_notebook()

    if cleanup_success:
        print("  Tips:")
        print("- Notebook er nå ryddet og klar for commit")
        print("- Du kan trygt pushe til GitHub")
        print("- Audio-spilleren vil fortsatt fungere når notebooken kjøres på␣
↪nytt")
        print("- Enhanced cleanup fjerner embedded audio/bilder automatisk")
    else:
        print("\\n  Cleanup feilet. Kjør manuelt:")
        print("  Cleanup failed. Run manually:")
        print("python scripts/enhanced_notebook_cleanup.py notebooks/
↪00_slektstraer_og_grafer.ipynb")
```

1 Slektstrær og Grafer - En Introduksjon

2 Family Trees and Graphs - An Introduction

Norsk | **English**

2.1 Velkommen til slektstre-prosjektet! / Welcome to the Family Tree Project!

Denne notebooken gir deg en grundig introduksjon til både slektstrær (genealogi) og grafteori, og hvordan disse to fagområdene kobles sammen i dette prosjektet.

This notebook provides you with a comprehensive introduction to both family trees (genealogy) and graph theory, and how these two fields are connected in this project.

2.1.1 Hva vil du lære? / What will you learn?

Norsk: - Hva slektstrær er og hvorfor de er viktige - Grunnleggende grafteori og hvordan den gjelder for slektstrær - Praktiske øvelser med både refleksjon og programmering - Hvordan NetworkX brukes til å bygge og analysere slektstrær

English: - What family trees are and why they matter - Basic graph theory and how it applies to family trees - Practical exercises with both reflection and programming - How NetworkX is used to build and analyze family trees

2.1.2 Forutsetninger / Prerequisites

Norsk: - Grunnleggende Python-kunnskap (anbefalt) - Ingen forkunnskap om grafteori nødvendig - Åpenhet for å lære nye konsepter

English: - Basic Python knowledge (recommended) - No prior graph theory knowledge required - Openness to learning new concepts

---

2.2 Del 1: Hva er slektstrær? / Part 1: What are Family Trees?

2.2.1 Slektstrær - En historisk oversikt

Et **slektstre** (også kalt **stamtre** eller **genealogi**) er en visuell representasjon av slektskap og familierelasjoner. Slektstrær har eksistert i tusenvis av år og har spilt en viktig rolle i menneskelig kultur og historie.

Hvorfor er slektstrær viktige?

1. **Personlig identitet**: Hjelper oss å forstå vår egen bakgrunn
2. **Historisk bevaring**: Bevarer familiens historie for fremtidige generasjoner
3. **Medisinsk informasjon**: Kan gi viktig informasjon om arvelige sykdommer
4. **Kulturell betydning**: Kobler oss til vår kulturelle og etniske bakgrunn
5. **Historisk forskning**: Hjelper historikere å forstå befolkningsbevegelser og sosiale strukturer

Typer slektstrær **1. Stamtavle (Pedigree Chart)** - Viser forfedre til en bestemt person - Tradisjonell tre-struktur - Brukes ofte i medisinsk kontekst

2. Etterkommertre (Descendant Tree) - Viser alle etterkommere fra en bestemt forfader - Nyttig for å se familiens utbredelse

3. Timeglass-visning (Hourglass Chart) - Viser både forfedre og etterkommere - Fokusperson i midten - Gir komplett oversikt

4. Vifte-diagram (Fan Chart) - Sirkulær visning av forfedre - Estetisk tiltalende - Populær i moderne genealogi-programmer

2.2.2 Family Trees - A Historical Overview

A **family tree** (also called **pedigree** or **genealogy**) is a visual representation of kinship and family relationships. Family trees have existed for thousands of years and have played an important role in human culture and history.

Why are family trees important?

1. **Personal identity**: Helps us understand our own background
2. **Historical preservation**: Preserves family history for future generations
3. **Medical information**: Can provide important information about hereditary diseases
4. **Cultural significance**: Connects us to our cultural and ethnic background
5. **Historical research**: Helps historians understand population movements and social structures

Types of family trees **1. Pedigree Chart** - Shows ancestors of a specific person - Traditional tree structure - Often used in medical context

2. Descendant Tree - Shows all descendants from a specific ancestor - Useful for seeing family spread

3. Hourglass Chart - Shows both ancestors and descendants - Focus person in the center - Provides complete overview

4. Fan Chart - Circular view of ancestors - Aesthetically pleasing - Popular in modern genealogy software

---

2.2.3 Refleksjonsoppgaver / Reflection Questions

Norsk: 1. Hvilke typer slektskap kjenner du til i din egen familie? 2. Hvor langt tilbake i tid kan du spore din familie? 3. Hvilke historier eller tradisjoner har blitt videreført i din familie? 4. Hvorfor tror du slektstrær er viktige for mennesker?

English: 1. What types of relationships do you know about in your own family? 2. How far back in time can you trace your family? 3. What stories or traditions have been passed down in your family? 4. Why do you think family trees are important to people?

2.3 Del 2: Grunnleggende grafteori / Part 2: Basic Graph Theory

2.3.1 Hva er en graf? / What is a Graph?

En **graf** i matematikk og informatikk er en samling av **noder** (også kalt **hjørner** eller **vertices**) som er koblet sammen med **kanter** (edges). Dette er et kraftig konsept som brukes i mange områder, inkludert slektstrær!

A graph** in mathematics and computer science is a collection of **nodes** (also called **vertices**) that are connected by **edges**. This is a powerful concept used in many areas, including family trees!**

Grunnleggende begreper / Basic Concepts **Norsk:** - **Node/Hjørne**: Et punkt i grafen (f.eks. en person) - **Kant**: En linje som kobler to noder (f.eks. en slektsrelasjon) - **Retted graf**: Kanter har retning (A → B) - **Ikke-retted graf**: Kanter har ingen retning (A B) - **Tre**: En spesiell type graf uten sykler

English: - **Node/Vertex**: A point in the graph (e.g., a person) - **Edge**: A line connecting two nodes (e.g., a family relationship) - **Directed graph**: Edges have direction (A → B) - **Undirected graph**: Edges have no direction (A B) - **Tree**: A special type of graph without cycles

2.3.2 Eksempler med NetworkX / Examples with NetworkX

La oss se på noen enkle eksempler:

[36]:
```
# Importer nødvendige biblioteker
import networkx as nx
import matplotlib.pyplot as plt
from datetime import date

# Importer slektstre-moduler (fungerer både lokalt og i Colab)
if IN_COLAB:
    # Bruk de modulene vi lastet inn i Colab-setup
    Person = slektstre_models.Person
    Gender = slektstre_models.Gender
    Slektstre = slektstre_tree.Slektstre
    load_from_yaml = slektstre_io.load_from_yaml
else:
    # Lokale imports
    from models import Person, Gender
    from tree import Slektstre
    from family_io import load_from_yaml

print(" Biblioteker importert!")
print(" Libraries imported!")
```

2.4 Eksempel 4: Bygg et enkelt slektstre som graf

2.5 Example 4: Build a simple family tree as graph

I dette eksemplet skal vi bygge et enkelt slektstre ved å bruke NetworkX-grafbiblioteket. Vi vil opprette noder for familiemedlemmer og knytte dem sammen med kanter som representerer familierelasjoner.

Læringsmål / Learning objectives: - Forstå hvordan slektstre kan representeres som grafer - Lære å bruke NetworkX for å bygge grafer - Se forskjellen mellom rettede og ikke-rettede kanter - Forstå hvordan familierelasjoner kan modelleres

Hva vi skal gjøre / What we will do: 1. Opprette en rettet graf med NetworkX 2. Legge til noder for familiemedlemmer 3. Koble sammen familierelasjoner med kanter 4. Visualisere slektstreet

2.6 Nabomatrise (Adjacency Matrix)

En nabomatrise er en matematisk representasjon av en graf som viser hvilke noder som er koblet sammen. I en nabomatrise A er A[i,j] = 1 hvis det finnes en kant fra node i til node j, og 0 hvis ikke.

Viktige egenskaper / Important properties: - **Symmetrisk for ikke-rettede grafer**: A[i,j] = A[j,i] - **Ikke-symmetrisk for rettede grafer**: A[i,j] kan være forskjellig fra A[j,i] - **Blandet graf**: Kombinasjon av rettede og ikke-rettede kanter - **Diagonal**: A[i,i] = 0 (ingen node kobler til seg selv)

I slektstre / In family trees: - Rettede kanter: Forelder → Barn (asymmetrisk) - Ikke-rettede kanter: Ekteskap (symmetrisk) - Resultatet blir en ikke-symmetrisk nabomatrise

2.7 Node-grader i blandede grafer

I blandede grafer (som slektstre) har vi både rettede og ikke-rettede kanter. Dette gir oss tre typer grader for hver node:

Matematiske definisjoner / Mathematical definitions: - **Grad (Degree)**: Totalt antall kanter som er koblet til noden - **Inn-grad (In-degree)**: Antall rettede kanter som peker INN til noden - **Ut-grad (Out-degree)**: Antall rettede kanter som peker UT fra noden

I slektstre / In family trees: - **Inn-grad**: Antall foreldre (vanligvis 0 eller 2) - **Ut-grad**: Antall barn - **Grad**: Antall foreldre + antall barn + antall ektefeller

Eksempel / Example: - En person med 2 foreldre, 3 barn og 1 ektefelle har: - Inn-grad: 2 - Ut-grad: 3
- Grad: 2 + 3 + 1 = 6

[37]:
```python
# Eksempel 1: Enkel urettet graf / Simple undirected graph
print("  Eksempel 1: Enkel urettet graf")
print("  Example 1: Simple undirected graph")

# Opprett en enkel graf
G = nx.Graph()
```

```python
# Legg til noder (5 totalt)
G.add_node("A", name="Alice")
G.add_node("B", name="Bob")
G.add_node("C", name="Charlie")
G.add_node("D", name="David")
G.add_node("E", name="Emma")

# Legg til kanter
G.add_edge("A", "B")
G.add_edge("B", "C")
G.add_edge("C", "D")
G.add_edge("D", "E")
G.add_edge("E", "A")  # Lager en sykel for å vise forskjell fra tre

# Visualiser med fiksert layout
plt.figure(figsize=(10, 8))

# Definer posisjoner manuelt for konsistent layout
pos = {
    "A": (0, 1),      # Alice øverst i midten
    "B": (-1, 0),     # Bob til venstre
    "C": (-0.5, -1),  # Charlie nederst til venstre
    "D": (0.5, -1),   # David nederst til høyre
    "E": (1, 0)       # Emma til høyre
}

nx.draw(G, pos, with_labels=True, node_color='lightblue',
        node_size=1000, font_size=16, font_weight='bold')
plt.title("Enkel urettet graf / Simple undirected graph")
plt.show()

print(f"Antall noder: {G.number_of_nodes()}")
print(f"Antall kanter: {G.number_of_edges()}")
print(f"Number of nodes: {G.number_of_nodes()}")
print(f"Number of edges: {G.number_of_edges()}")

# Vis alle noder og deres navn
print(f"\nNoder i grafen:")
print(f"Nodes in the graph:")
for node in G.nodes():
    print(f"  {node}: {G.nodes[node]['name']}")
```

2.8 Avstands-matrise (Distance Matrix)

En avstands-matrise viser den korteste stien mellom alle par av noder i en graf. Avstanden er antall kanter man må gå for å komme fra en node til en annen.

Viktige egenskaper / Important properties: - **Avstand**: Antall kanter i korteste sti mellom

to noder - **Diagonal**: Avstand fra node til seg selv er alltid 0 - **Symmetrisk**: Avstanden fra A til B er samme som fra B til A - **Uendelig**: Hvis det ikke finnes sti mellom to noder

I slektstre / In family trees: - **Biologisk beslektet**: Forelder-barn, søsken (deler blod) - **Ikke-biologisk beslektet**: Ektefeller, svigerfamilie - **Avstand 1**: Direkte relasjon (forelder-barn, ektefelle) - **Avstand 2**: Besteforeldre-barn, svigerforeldre

Eksempel / Example: - Far til barn: avstand 1 (direkte forelder-barn) - Bestefar til barnebarn: avstand 2 (via forelder)

[38]:
```
# Eksempel 2: Rettet graf / Directed graph
print("  Eksempel 2: Rettet graf")
print("  Example 2: Directed graph")

# Opprett en rettet graf
D = nx.DiGraph()

# Legg til noder
D.add_node("Parent", name="Forelder")
D.add_node("Child1", name="Barn1")
D.add_node("Child2", name="Barn2")

# Legg til rettede kanter (forelder → barn)
D.add_edge("Parent", "Child1")
D.add_edge("Parent", "Child2")

# Visualiser med manuell posisjonering
plt.figure(figsize=(8, 6))

# Definer posisjoner manuelt: Forelder øverst, Barn1 og Barn2 på samme linje␣
 ↪nedenfor
pos = {
    "Parent": (0, 1),     # Forelder øverst i midten
    "Child1": (-0.5, 0), # Barn1 til venstre nedenfor
    "Child2": (0.5, 0)   # Barn2 til høyre nedenfor
}

# Tegn noder
nx.draw_networkx_nodes(D, pos, node_color='lightgreen',
                       node_size=1000, alpha=0.7)

# Tegn kanter
nx.draw_networkx_edges(D, pos, arrows=True, arrowsize=20,
                       edge_color='black', width=2)

# Legg til etiketter med norske navn
labels = {node: D.nodes[node]['name'] for node in D.nodes()}
nx.draw_networkx_labels(D, pos, labels, font_size=16, font_weight='bold')
```

```
plt.title("Rettet graf (forelder-barn) / Directed graph (parent-child)")
plt.axis('off')
plt.show()

print(f"Antall noder: {D.number_of_nodes()}")
print(f"Antall kanter: {D.number_of_edges()}")
print(f"Number of nodes: {D.number_of_nodes()}")
print(f"Number of edges: {D.number_of_edges()}")
```

2.9 Similaritetsmål i slektstre

Similaritetsmål måler hvor like to personer er basert på forskjellige attributter. Dette er forskjellig fra avstandsmatriser som bare måler strukturell avstand.

Typer similaritetsmål / Types of similarity measures: - **Kjønn**: Like kjønn = høy similarity, forskjellig kjønn = lav similarity - **Alder**: Mindre aldersforskjell = høyere similarity - **Geografisk**: Samme bosted = høy similarity - **Yrke**: Samme yrke = høy similarity - **Navn**: Likhet i navn (Levenshtein distance)

Kombinert similarity / Combined similarity: Vi kan kombinere flere similarity-mål ved å gi dem vekter og beregne et vektet gjennomsnitt.

Anvendelse / Applications: - Finne personer med lik bakgrunn - Identifisere potensielle slektninger - Gruppere familiemedlemmer etter likhet - Analysere familiemønstre

2.9.1 Spesielle typer grafer / Special Types of Graphs

1. Tre (Tree) Et **tre** er en spesiell type graf som: - Har ingen sykler (ikke kan gå i sirkel) - Er sammenkoblet (alle noder kan nås fra hverandre) - Har nøyaktig n-1 kanter for n noder

A tree** is a special type of graph that:** - Has no cycles (cannot go in circles) - Is connected (all nodes can be reached from each other) - Has exactly n-1 edges for n nodes

2. Sykler (Cycles) En **sykel** er når du kan gå fra en node og komme tilbake til samme node. I slektstrær vil dette være umulig (en person kan ikke være sin egen forfader).

A cycle** is when you can go from a node and return to the same node. In family trees this would be impossible (a person cannot be their own ancestor).**

3. Grad (Degree) **Grad** er antall kanter som er koblet til en node: - **Inngrad**: Antall kanter som kommer inn til noden - **Utgrad**: Antall kanter som går ut fra noden

Degree is the number of edges connected to a node: **- In-degree: Number of edges coming into the node -** Out-degree**: Number of edges going out from the node

2.10 Slektstre med attributter

I denne visualiseringen viser vi slektstreet med alle attributtene vi har definert. Dette gir oss en komplett oversikt over både strukturen og egenskapene til familiemedlemmene.

Visualiseringselementer / Visualization elements: - **Node-farger**: Blå for menn, rød for kvinner - **Node-størrelse**: Proporsjonal med alder (eldre = større) - **Node-labels**: Fornavn for bedre lesbarhet - **Attributt-bokser**: Viser kjønn, alder, bosted og yrke - **Kant-typer**: Heltrukne for forelder-barn, stiplede for ekteskap - **Legend**: Forklarer alle elementer

Insights vi kan få / Insights we can gain: - Se aldersfordeling i familien - Identifisere geografiske mønstre - Se yrkesfordeling - Forstå familierelasjoner visuelt

[39]:
```
# Eksempel 3: Tre vs. Graf med sykler / Tree vs. Graph with cycles
print("  Eksempel 3: Tre vs. Graf med sykler")
print("  Example 3: Tree vs. Graph with cycles")

# Opprett et tre
tree = nx.Graph()
tree.add_edges_from([(1, 2), (1, 3), (2, 4), (2, 5)])

# Opprett en graf med sykler
cycle_graph = nx.Graph()
cycle_graph.add_edges_from([(1, 2), (2, 3), (3, 4), (4, 1)])

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Tegn treet med fiksert layout
pos1 = {
    1: (0, 1),      # Node 1 øverst i midten
    2: (0, 0),      # Node 2 i midten
    3: (-0.8, 0.5), # Node 3 til venstre
    4: (-0.5, -0.8), # Node 4 nederst til venstre
    5: (0.5, -0.8)  # Node 5 nederst til høyre
}
nx.draw(tree, pos1, with_labels=True, node_color='lightblue',
        node_size=1000, font_size=16, font_weight='bold', ax=ax1)
ax1.set_title("Tre (ingen sykler) / Tree (no cycles)")

# Tegn grafen med sykler med fiksert layout
pos2 = {
    1: (0, 1),      # Node 1 øverst
    2: (1, 0),      # Node 2 til høyre
    3: (0, -1),     # Node 3 nederst
    4: (-1, 0)      # Node 4 til venstre
}
nx.draw(cycle_graph, pos2, with_labels=True, node_color='lightcoral',
        node_size=1000, font_size=16, font_weight='bold', ax=ax2)
ax2.set_title("Graf med sykler / Graph with cycles")

plt.tight_layout()
plt.show()
```

```
print(f"Tre - Er tre: {nx.is_tree(tree)}")
print(f"Sykelgraf - Er tre: {nx.is_tree(cycle_graph)}")
print(f"Tree - Is tree: {nx.is_tree(tree)}")
print(f"Cycle graph - Is tree: {nx.is_tree(cycle_graph)}")
```

2.11 Visualisering av alle similarity-matriser

I denne seksjonen visualiserer vi alle similarity-matrisene vi har beregnet som heatmaps. Dette gir oss en omfattende oversikt over hvordan familiemedlemmene er like hverandre på forskjellige måter.

Hva vi visualiserer / What we visualize: - **Gender Similarity**: Kjønnsbasert similarity - **Age Similarity**: Aldersbasert similarity
- **Location Similarity**: Geografisk similarity - **Profession Similarity**: Yrkesbasert similarity - **Name Similarity**: Navnebasert similarity - **Combined Similarity**: Kombinert similarity-score

Analytiske insights / Analytical insights: - Identifiser par med høyest similarity - Se korrelasjoner mellom forskjellige similarity-mål - Forstå hvilke attributter som påvirker similarity mest - Finne interessante mønstre i familien

Heatmap-tolkning / Heatmap interpretation: - **Lys farge**: Høy similarity (mer lik) - **Mørk farge**: Lav similarity (mindre lik) - **Diagonal**: Alltid 1.0 (person lik seg selv)

2.12 Del 3: Slektstrær SOM grafer / Part 3: Family Trees AS Graphs

2.12.1 Kobling mellom slektstrær og grafteori / Connection between Family Trees and Graph Theory

Nå skal vi se hvordan slektstrær kan representeres som grafer:

Now we'll see how family trees can be represented as graphs:

Mapping / Kartlegging

Slektstre / Family Tree	Graf / Graph
Person / Person	Node / Node
Slektsrelasjon / Family relationship	Kant / Edge
Forelder-barn / Parent-child	Retted kant / Directed edge
Ekteskap / Marriage	Ikke-retted kant / Undirected edge
Generasjon / Generation	Avstand fra rot / Distance from root
Slektskap / Kinship	Sti mellom noder / Path between nodes

Hvorfor NetworkX er perfekt for slektstrær / Why NetworkX is perfect for family trees

1. **Kraftige algoritmer**: NetworkX har innebygde algoritmer for å finne stier, beregne avstander, og analysere grafer
2. **Visualisering**: Enkelt å lage visuelle representasjoner

3. **Fleksibilitet**: Støtter både rettede og ikke-rettede grafer
4. **Skalerbarhet**: Kan håndtere store slektstrær
5. **Analyse**: Kan beregne komplekse slektskap automatisk

English: 1. **Powerful algorithms**: NetworkX has built-in algorithms for finding paths, calculating distances, and analyzing graphs 2. **Visualization**: Easy to create visual representations 3. **Flexibility**: Supports both directed and undirected graphs 4. **Scalability**: Can handle large family trees 5. **Analysis**: Can calculate complex relationships automatically

[40]:
```
# Eksempel 4: Bygg et enkelt slektstre som graf / Build a simple family tree as␣
↪graph
print("  Eksempel 4: Bygg et enkelt slektstre som graf")
print("  Example 4: Build a simple family tree as graph")

# Opprett et retnet graf for slektstreet
family_graph = nx.DiGraph()

# Legg til personer som noder
family_graph.add_node("Bestefar", name="Erik Lundervold", generation=1)
family_graph.add_node("Bestemor", name="Ingrid Hansen", generation=1)
family_graph.add_node("Far", name="Arvid Lundervold", generation=2)
family_graph.add_node("Mor", name="Anna Pedersen", generation=2)
family_graph.add_node("Barn1", name="Lars Lundervold", generation=3)
family_graph.add_node("Barn2", name="Kari Lundervold", generation=3)

# Legg til forelder-barn relasjoner (retnede kanter)
family_graph.add_edge("Bestefar", "Far", relation="parent-child")
family_graph.add_edge("Bestemor", "Far", relation="parent-child")
family_graph.add_edge("Far", "Barn1", relation="parent-child")
family_graph.add_edge("Far", "Barn2", relation="parent-child")
family_graph.add_edge("Mor", "Barn1", relation="parent-child")
family_graph.add_edge("Mor", "Barn2", relation="parent-child")

# Legg til ekteskap (uretnede kanter)
family_graph.add_edge("Bestefar", "Bestemor", relation="marriage")
family_graph.add_edge("Far", "Mor", relation="marriage")

# Visualiser med fiksert slektstre-layout
plt.figure(figsize=(12, 8))

# Definer fiksert posisjonering som et typisk slektstre
pos = {
    "Bestefar": (0, 2),      # Øverst til venstre
    "Bestemor": (2, 2),      # Øverst til høyre
    "Far": (0, 1),           # Midten til venstre
    "Mor": (2, 1),           # Midten til høyre
    "Barn1": (0, 0),         # Nederst til venstre
    "Barn2": (2, 0)          # Nederst til høyre
```

```
}

# Tegn noder
nx.draw_networkx_nodes(family_graph, pos, node_color='lightblue',
                       node_size=2000, alpha=0.7)

# Tegn kanter med forskjellige farger
parent_child_edges = [(u, v) for u, v, d in family_graph.edges(data=True)
                      if d.get('relation') == 'parent-child']
marriage_edges = [(u, v) for u, v, d in family_graph.edges(data=True)
                  if d.get('relation') == 'marriage']

nx.draw_networkx_edges(family_graph, pos, edgelist=parent_child_edges,
                       edge_color='black', arrows=True, arrowsize=20, width=2)
nx.draw_networkx_edges(family_graph, pos, edgelist=marriage_edges,
                       edge_color='red', arrows=False, width=3, style='dashed')

# Legg til etiketter
labels = {node: family_graph.nodes[node]['name'] for node in family_graph.
 ↪nodes()}
nx.draw_networkx_labels(family_graph, pos, labels, font_size=10,␣
 ↪font_weight='bold')

plt.title("Slektstre som graf / Family tree as graph")
plt.axis('off')
plt.show()

print(f"Antall personer: {family_graph.number_of_nodes()}")
print(f"Antall relasjoner: {family_graph.number_of_edges()}")
print(f"Number of people: {family_graph.number_of_nodes()}")
print(f"Number of relationships: {family_graph.number_of_edges()}")

# Vis familierelasjoner som tabell
print(f"\n  Familierelasjoner / Family relationships:")
print(f"{'='*60}")
print(f"{'Fra (From)':<20} {'Til (To)':<20} {'Relasjon (Relation)':<20}")
print(f"{'='*60}")

for u, v, data in family_graph.edges(data=True):
    from_name = family_graph.nodes[u]['name']
    to_name = family_graph.nodes[v]['name']
    relation = data.get('relation', 'unknown')

    # Oversett relasjon til norsk/engelsk
    if relation == 'parent-child':
        rel_norsk = "Forelder-barn"
        rel_engelsk = "Parent-child"
```

```
    elif relation == 'marriage':
        rel_norsk = "Ekteskap"
        rel_engelsk = "Marriage"
    else:
        rel_norsk = relation
        rel_engelsk = relation

    print(f"{from_name:<20} {to_name:<20} {rel_norsk}/{rel_engelsk:<20}")

print(f"{'='*60}")

# Vis generasjonsinformasjon
print(f"\n   Generasjonsoversikt / Generation overview:")
generations = {}
for node, data in family_graph.nodes(data=True):
    gen = data.get('generation', 'unknown')
    if gen not in generations:
        generations[gen] = []
    generations[gen].append(data['name'])

for gen in sorted(generations.keys()):
    print(f"Generasjon {gen}: {', '.join(generations[gen])}")
```

[41]:
```
# Nabomatrise for slektstreet / Adjacency matrix for the family tree
print("  Nabomatrise / Adjacency Matrix")
print("  Norsk forklaring:")
print("En nabomatrise er en kvadratisk matrise som representerer en graf.")
print("Hver rad og kolonne representerer en node, og verdien 1/0 viser om␣
 ↪nodene er koblet.")
print("Dette er en alternativ og likeverdig representasjon av grafen.")
print("")
print("  English explanation:")
print("An adjacency matrix is a square matrix that represents a graph.")
print("Each row and column represents a node, and the value 1/0 shows if nodes␣
 ↪are connected.")
print("This is an alternative and equivalent representation of the graph.")
print("")

# Opprett nabomatrise
nodes = list(family_graph.nodes())
n = len(nodes)

print("  6x6 Nabomatrise (1 = koblet, 0 = ikke koblet):")
print("  6x6 Adjacency matrix (1 = connected, 0 = not connected):")
print("")

# Lag header med fornavn
```

```
header = f"{'Node':<20}"
for node in nodes:
    first_name = family_graph.nodes[node]['name'].split()[0]  # Første ord␣
 ↪(fornavn)
    header += f"{first_name:<12}"
print(header)
print("-" * (20 + 12 * len(nodes)))

# Beregn og vis nabomatrisen
for i, node1 in enumerate(nodes):
    first_name1 = family_graph.nodes[node1]['name'].split()[0]  # Første ord␣
 ↪(fornavn)
    row = f"{first_name1:<20}"
    for j, node2 in enumerate(nodes):
        if i == j:
            # Diagonal - samme node (ikke koblet til seg selv)
            adjacency = 0
        else:
            # Sjekk om det finnes en kant mellom nodene
            if family_graph.has_edge(node1, node2):
                adjacency = 1
            else:
                adjacency = 0

        row += f"{adjacency:<12}"
    print(row)

print("")
print("  Forklaring av verdier / Explanation of values:")
print("• 1: Det finnes en kant mellom nodene / There is an edge between the␣
 ↪nodes")
print("• 0: Ingen kant mellom nodene / No edge between the nodes")
print("• Diagonal: Alltid 0 (ingen node kobler til seg selv) / Always 0 (no␣
 ↪self-loops)")
print("• Matrisen er IKKE symmetrisk på grunn av retnede og ikke-rettede␣
 ↪kanter")
print("• The matrix is NOT symmetric due to directed and undirected edges")
print("")

# Vis familierelasjoner med spesifikke titler
print("   Familierelasjoner / Family relationships:")
print("")

# Definer familierelasjoner
family_relations = {
    ("Bestefar", "Far"): "farfar",
    ("Bestemor", "Far"): "farmor",
```

```
    ("Far", "Barn1"): "far",
    ("Far", "Barn2"): "far",
    ("Mor", "Barn1"): "mor",
    ("Mor", "Barn2"): "mor",
    ("Bestefar", "Bestemor"): "ektefelle",
    ("Far", "Mor"): "ektefelle"
}

print("  Norske familierelasjoner:")
for (u, v), relation in family_relations.items():
    u_name = family_graph.nodes[u]['name'].split()[0]
    v_name = family_graph.nodes[v]['name'].split()[0]
    print(f"  {u_name} er {relation} til {v_name}")

print("")
print("  English family relationships:")
english_relations = {
    ("Bestefar", "Far"): "grandfather",
    ("Bestemor", "Far"): "grandmother",
    ("Far", "Barn1"): "father",
    ("Far", "Barn2"): "father",
    ("Mor", "Barn1"): "mother",
    ("Mor", "Barn2"): "mother",
    ("Bestefar", "Bestemor"): "spouse",
    ("Far", "Mor"): "spouse"
}

for (u, v), relation in english_relations.items():
    u_name = family_graph.nodes[u]['name'].split()[0]
    v_name = family_graph.nodes[v]['name'].split()[0]
    print(f"  {u_name} is {relation} to {v_name}")

print("")
print("="*60)
print("  LEGEND FOR GRAF-VISUALISERING / LEGEND FOR GRAPH VISUALIZATION")
print("="*60)
print("  Norsk:")
print("• Heltrukne kanter (→): Forelder-barn relasjoner (rettede kanter)")
print("• Stiplede kanter (---): Ekteskap (ikke-rettede kanter)")
print("• Piler (→): Viser retning fra forelder til barn")
print("• Ingen piler (---): Ekteskap er gjensidig")
print("• Nabomatrisen er IKKE symmetrisk på grunn av kanttyper")
print("")
print("  English:")
print("• Solid edges (→): Parent-child relationships (directed edges)")
print("• Dashed edges (---): Marriages (undirected edges)")
print("• Arrows (→): Show direction from parent to child")
```

```
print("• No arrows (---): Marriage is mutual")
print("• The adjacency matrix is NOT symmetric due to edge types")
print("")

print("="*60)
print("  MATEMATISK FORKLARING / MATHEMATICAL EXPLANATION")
print("="*60)
print("  Norsk:")
print("• Nabomatrise: A[i,j] = 1 hvis det finnes kant fra node i til node j,␣
 ↪ellers 0")
print("• Rettede kanter: A[i,j] kan være forskjellig fra A[j,i]")
print("• Ikke-rettede kanter: A[i,j] = A[j,i] (symmetrisk)")
print("• Blandet graf: Kombinasjon av rettede og ikke-rettede kanter")
print("• Nabomatrise og graf er likeverdige representasjoner")
print("")
print("  English:")
print("• Adjacency matrix: A[i,j] = 1 if there is an edge from node i to node␣
 ↪j, else 0")
print("• Directed edges: A[i,j] may differ from A[j,i]")
print("• Undirected edges: A[i,j] = A[j,i] (symmetric)")
print("• Mixed graph: Combination of directed and undirected edges")
print("• Adjacency matrix and graph are equivalent representations")
```

[42]:
```
# Node-orden, inn-orden og ut-orden / Node degree, in-degree and out-degree
print("  Node-orden i blandede grafer / Node degree in mixed graphs")
print("  Norsk forklaring:")
print("Node-orden er antall kanter som er koblet til en node.")
print("I blandede grafer (som slektstrær) har vi både retnede og uretnede␣
 ↪kanter.")
print("Vi skiller mellom inn-orden (kanter som kommer inn) og ut-orden (kanter␣
 ↪som går ut).")
print("")
print("  English explanation:")
print("Node degree is the number of edges connected to a node.")
print("In mixed graphs (like family trees) we have both directed and undirected␣
 ↪edges.")
print("We distinguish between in-degree (edges coming in) and out-degree (edges␣
 ↪going out).")
print("")

# Beregn node-ordener for hver node
print("  Node-orden for hver person / Node degree for each person:")
print("")

for node in family_graph.nodes():
    name = family_graph.nodes[node]['name']
    first_name = name.split()[0]
```

```
    # Beregn forskjellige ordener
    total_degree = family_graph.degree(node)  # Totalt antall kanter
    in_degree = family_graph.in_degree(node)   # Antall kanter som kommer inn
    out_degree = family_graph.out_degree(node) # Antall kanter som går ut

    print(f"  {first_name} ({name}):")
    print(f"    Totalt antall kanter: {total_degree}")
    print(f"    Total number of edges: {total_degree}")
    print(f"    Inn-orden (kanter inn): {in_degree}")
    print(f"    In-degree (edges in): {in_degree}")
    print(f"    Ut-orden (kanter ut): {out_degree}")
    print(f"    Out-degree (edges out): {out_degree}")
    print("")

# Vis detaljert analyse av kanttyper
print("  Detaljert analyse av kanttyper / Detailed analysis of edge types:")
print("")

for node in family_graph.nodes():
    name = family_graph.nodes[node]['name']
    first_name = name.split()[0]

    # Analyser hver kanttype
    parent_child_in = 0
    parent_child_out = 0
    marriage_edges = 0

    # Sjekk alle kanter til denne noden
    for neighbor in family_graph.neighbors(node):
        edge_data = family_graph.get_edge_data(node, neighbor)
        if edge_data and edge_data.get('relation') == 'parent-child':
            # Sjekk retning
            if family_graph.has_edge(node, neighbor):
                parent_child_out += 1
            if family_graph.has_edge(neighbor, node):
                parent_child_in += 1
        elif edge_data and edge_data.get('relation') == 'marriage':
            marriage_edges += 1

    print(f"  {first_name}:")
    print(f"    Forelder-barn (ut): {parent_child_out} kanter")
    print(f"    Parent-child (out): {parent_child_out} edges")
    print(f"    Forelder-barn (inn): {parent_child_in} kanter")
    print(f"    Parent-child (in): {parent_child_in} edges")
    print(f"    Ekteskap: {marriage_edges} kanter")
    print(f"    Marriage: {marriage_edges} edges")
```

```
    print("")

print("="*60)
print("  MATEMATISK DEFINISJONER / MATHEMATICAL DEFINITIONS")
print("="*60)
print("  Norsk:")
print("• Node-orden (degree): Totalt antall kanter koblet til en node")
print("• Inn-orden (in-degree): Antall kanter som kommer inn til noden")
print("• Ut-orden (out-degree): Antall kanter som går ut fra noden")
print("• For ikke-rettede kanter: Teller som både inn og ut")
print("• For rettede kanter: Teller kun i én retning")
print("• Blandet graf: Kombinasjon av begge kanttyper")
print("")
print("  English:")
print("• Node degree: Total number of edges connected to a node")
print("• In-degree: Number of edges coming into the node")
print("• Out-degree: Number of edges going out from the node")
print("• For undirected edges: Counts as both in and out")
print("• For directed edges: Counts only in one direction")
print("• Mixed graph: Combination of both edge types")
print("")

# Vis eksempler på hvordan dette gjelder for slektstrær
print("  Eksempler fra slektstreet / Examples from the family tree:")
print("")
print("  Norsk:")
print("• Bestefar: Ut-orden 2 (til Far og Bestemor), Inn-orden 1 (fra␣
 ↪Bestemor)")
print("• Far: Ut-orden 2 (til Barn1 og Barn2), Inn-orden 3 (fra Bestefar,␣
 ↪Bestemor, Mor)")
print("• Barn1: Ut-orden 0, Inn-orden 2 (fra Far og Mor)")
print("")
print("  English:")
print("• Grandfather: Out-degree 2 (to Father and Grandmother), In-degree 1␣
 ↪(from Grandmother)")
print("• Father: Out-degree 2 (to Child1 and Child2), In-degree 3 (from␣
 ↪Grandfather, Grandmother, Mother)")
print("• Child1: Out-degree 0, In-degree 2 (from Father and Mother)")
```

[43]:
```
# Avstands-matrise for slektstreet / Distance matrix for the family tree
print("  Avstands-matrise / Distance Matrix")
print("  Norsk forklaring:")
print("En avstands-matrise viser korteste sti-lengde mellom alle par av noder i␣
 ↪grafen.")
print("VIKTIG: Vi skiller mellom biologisk beslektede og ikke-biologisk␣
 ↪beslektede relasjoner.")
```

```
print("Forelder-barn og søsken er biologisk beslektet, ekteskap er som regel␣
 ↪ikke biologisk beslektet.")
print("")
print("  English explanation:")
print("A distance matrix shows the shortest path length between all pairs of␣
 ↪nodes in the graph.")
print("IMPORTANT: We distinguish between biologically related and␣
 ↪non-biologically related relationships.")
print("Parent-child and siblings are biologically related, marriages are␣
 ↪usually not biologically related.")
print("")

# Beregn korteste sti-lengder mellom alle noder
nodes = list(family_graph.nodes())

print("  6x6 Avstands-matrise (korteste sti-lengde):")
print("  6x6 Distance matrix (shortest path length):")
print("")

# Lag header med fornavn
header = f"{'Node':<15}"
for node in nodes:
    first_name = family_graph.nodes[node]['name'].split()[0]  # Første ord␣
 ↪(fornavn)
    header += f"{first_name:<12}"
print(header)
print("-" * (15 + 12 * len(nodes)))

# Beregn og vis matrisen
for i, node1 in enumerate(nodes):
    first_name1 = family_graph.nodes[node1]['name'].split()[0]  # Første ord␣
 ↪(fornavn)
    row = f"{first_name1:<15}"
    for j, node2 in enumerate(nodes):
        if i == j:
            # Diagonal - samme node
            distance = 0
        else:
            try:
                # Finn korteste sti-lengde
                distance = nx.shortest_path_length(family_graph, node1, node2)
            except nx.NetworkXNoPath:
                # Ingen sti funnet
                distance = float('inf')

        if distance == float('inf'):
```

```
            row += f"{'∞':<12}"
        else:
            row += f"{distance:<12}"
    print(row)

print("")
print(" Forklaring av verdier / Explanation of values:")
print("• 0: Samme person / Same person")
print("• 1: Direkte koblet (forelder-barn eller ekteskap) / Directly connected")
print("• 2: Koblet via én mellomperson / Connected via one intermediate person")
print("• 3: Koblet via to mellompersoner / Connected via two intermediate␣
 ↪persons")
print("• ∞: Ingen sti funnet / No path found")
print("")

# Analyser biologisk vs ikke-biologisk beslektede relasjoner
print(" Biologisk vs ikke-biologisk beslektede relasjoner:")
print(" Biologically vs non-biologically related relationships:")
print("")

# Definer biologisk beslektede relasjoner
biological_relations = []
non_biological_relations = []

for u, v, data in family_graph.edges(data=True):
    relation = data.get('relation', 'unknown')
    u_name = family_graph.nodes[u]['name'].split()[0]
    v_name = family_graph.nodes[v]['name'].split()[0]

    if relation == 'parent-child':
        biological_relations.append((u_name, v_name, "forelder-barn"))
    elif relation == 'marriage':
        non_biological_relations.append((u_name, v_name, "ekteskap"))

print(" Biologisk beslektede (blodsrelasjoner):")
print(" Biologically related (blood relationships):")
for u_name, v_name, rel_type in biological_relations:
    print(f" {u_name} {v_name} ({rel_type})")

print("")
print(" Ikke-biologisk beslektede (ikke blodsrelasjoner):")
print(" Non-biologically related (non-blood relationships):")
for u_name, v_name, rel_type in non_biological_relations:
    print(f" {u_name} {v_name} ({rel_type})")

print("")
print(" Søsken-relasjoner (implisitte, biologisk beslektede):")
```

```
print("  Sibling relationships (implicit, biologically related):")
# Finn søsken-par
siblings = []
for node1 in family_graph.nodes():
    for node2 in family_graph.nodes():
        if node1 != node2:
            # Sjekk om de har samme foreldre
            parents1 = set(family_graph.predecessors(node1))
            parents2 = set(family_graph.predecessors(node2))
            if parents1 and parents2 and parents1 == parents2:
                name1 = family_graph.nodes[node1]['name'].split()[0]
                name2 = family_graph.nodes[node2]['name'].split()[0]
                if (name2, name1, "søsken") not in siblings:
                    siblings.append((name1, name2, "søsken"))

for name1, name2, rel_type in siblings:
    print(f"  {name1}  {name2} ({rel_type})")

print("")
print("="*60)
print("  BIOLOGISK OG MATEMATISK FORKLARING / BIOLOGICAL AND MATHEMATICAL␣
 ↪EXPLANATION")
print("="*60)
print("  Norsk:")
print("• Biologisk beslektede: Blodsrelasjoner (forelder-barn, søsken,␣
 ↪besteforeldre-barn)")
print("• Ikke-biologisk beslektede: Ikke blodsrelasjoner (ektefeller,␣
 ↪svigerfamilie)")
print("• Avstands-matrise: Viser strukturell avstand, ikke biologisk avstand")
print("• Sti-lengde 1: Direkte kobling (biologisk eller ikke-biologisk)")
print("• Sti-lengde 2: Via én mellomperson (kan være biologisk eller ikke)")
print("")
print("  English:")
print("• Biologically related: Blood relationships (parent-child, siblings,␣
 ↪grandparent-grandchild)")
print("• Non-biologically related: Non-blood relationships (spouses, in-laws)")
print("• Distance matrix: Shows structural distance, not biological distance")
print("• Path length 1: Direct connection (biological or non-biological)")
print("• Path length 2: Via one intermediate person (can be biological or␣
 ↪non-biological)")
print("")

# Vis noen interessante eksempler
print("  Eksempler på korteste stier / Examples of shortest paths:")
print("")

# Finn noen interessante stier
```

```
interesting_pairs = [
    ("Bestefar", "Barn1"),
    ("Bestemor", "Barn2"),
    ("Far", "Mor"),
    ("Barn1", "Barn2")
]

for start, end in interesting_pairs:
    try:
        path = nx.shortest_path(family_graph, start, end)
        path_names = [family_graph.nodes[node]['name'].split()[0] for node in␣
 ↪path]
        distance = len(path) - 1

        # Analyser om stien går gjennom biologisk eller ikke-biologisk␣
 ↪beslektede
        biological_steps = 0
        non_biological_steps = 0

        for i in range(len(path) - 1):
            u, v = path[i], path[i + 1]
            edge_data = family_graph.get_edge_data(u, v)
            if edge_data and edge_data.get('relation') == 'parent-child':
                biological_steps += 1
            elif edge_data and edge_data.get('relation') == 'marriage':
                non_biological_steps += 1

        print(f"Sti fra {family_graph.nodes[start]['name'].split()[0]} til␣
 ↪{family_graph.nodes[end]['name'].split()[0]}:")
        print(f"Path from {family_graph.nodes[start]['name'].split()[0]} to␣
 ↪{family_graph.nodes[end]['name'].split()[0]}:")
        print(f"  {' → '.join(path_names)} (lengde: {distance})")
        print(f"  {' → '.join(path_names)} (length: {distance})")
        print(f"  Biologisk beslektede steg: {biological_steps}")
        print(f"  Biologically related steps: {biological_steps}")
        print(f"  Ikke-biologisk beslektede steg: {non_biological_steps}")
        print(f"  Non-biologically related steps: {non_biological_steps}")
        print("")
    except nx.NetworkXNoPath:
        print(f"Ingen sti fra {start} til {end}")
        print(f"No path from {start} to {end}")
        print("")
```

[44]:
```
# Similaritets-mål i slektstrær / Similarity measures in family trees
print("  Similaritets-mål i slektstrær / Similarity measures in family trees")
print("  Norsk forklaring:")
```

```
print("I slektstrær kan vi definere similaritets-mål basert på forskjellige␣
 ↪attributter.")
print("Dette er forskjellig fra avstands-matrisen som kun viser strukturell␣
 ↪avstand.")
print("Similaritets-mål kan være knyttet til kjønn, alder, bosted, yrke,␣
 ↪navnelikhet, etc.")
print("")
print("  English explanation:")
print("In family trees we can define similarity measures based on different␣
 ↪attributes.")
print("This is different from the distance matrix which only shows structural␣
 ↪distance.")
print("Similarity measures can be related to gender, age, location, profession,␣
 ↪name similarity, etc.")
print("")

# Legg til attributter til familien for å demonstrere similaritets-mål
print("  Legger til attributter for similaritets-analyse:")
print("  Adding attributes for similarity analysis:")
print("")

# Oppdater familien med attributter
family_attributes = {
    "Bestefar": {"gender": "male", "age": 75, "location": "Oslo", "profession":␣
 ↪"ingeniør", "name": "Erik Lundervold"},
    "Bestemor": {"gender": "female", "age": 72, "location": "Oslo",␣
 ↪"profession": "lærer", "name": "Ingrid Hansen"},
    "Far": {"gender": "male", "age": 45, "location": "Bergen", "profession":␣
 ↪"programmerer", "name": "Arvid Lundervold"},
    "Mor": {"gender": "female", "age": 42, "location": "Bergen", "profession":␣
 ↪"lege", "name": "Anna Pedersen"},
    "Barn1": {"gender": "male", "age": 18, "location": "Bergen", "profession":␣
 ↪"student", "name": "Lars Lundervold"},
    "Barn2": {"gender": "female", "age": 16, "location": "Bergen", "profession":
 ↪ "student", "name": "Kari Lundervold"}
}

# Legg til attributter til grafen
for node, attrs in family_attributes.items():
    for attr, value in attrs.items():
        family_graph.nodes[node][attr] = value

print("  Attributter lagt til:")
for node, attrs in family_attributes.items():
    name = attrs["name"].split()[0]
```

```
    print(f"  {name}: {attrs['gender']}, {attrs['age']} år,␣
 ↪{attrs['location']}, {attrs['profession']}")

print("")

# Definer similaritets-funksjoner
def gender_similarity(person1, person2):
    """Kjønnssimilaritet: 1 hvis samme kjønn, 0 hvis forskjellig"""
    return 1 if person1["gender"] == person2["gender"] else 0

def age_similarity(person1, person2):
    """Alderssimilaritet: Jo nærmere alder, jo høyere similaritet"""
    age_diff = abs(person1["age"] - person2["age"])
    max_age_diff = 60  # Normaliser til 0-1 skala
    return max(0, 1 - (age_diff / max_age_diff))

def location_similarity(person1, person2):
    """Bostedssimilaritet: 1 hvis samme sted, 0 hvis forskjellig"""
    return 1 if person1["location"] == person2["location"] else 0

def profession_similarity(person1, person2):
    """Yrkessimilaritet: 1 hvis samme yrke, 0 hvis forskjellig"""
    return 1 if person1["profession"] == person2["profession"] else 0

def name_similarity(person1, person2):
    """Navnelikhet: Sjekker om de deler etternavn"""
    name1_parts = person1["name"].split()
    name2_parts = person2["name"].split()
    # Sjekk om de deler etternavn
    return 1 if name1_parts[-1] == name2_parts[-1] else 0

# Beregn similaritets-matrise for hver attributt
similarity_measures = {
    "Kjønn": gender_similarity,
    "Alder": age_similarity,
    "Bosted": location_similarity,
    "Yrke": profession_similarity,
    "Navnelikhet": name_similarity
}

print(" Similaritets-matriser for forskjellige attributter:")
print(" Similarity matrices for different attributes:")
print("")

for measure_name, similarity_func in similarity_measures.items():
    print(f"  {measure_name} / {measure_name}:")
    print(f"  {measure_name}:")
```

```
    # Lag header
    header = f"{'Node':<15}"
    for node in family_graph.nodes():
        first_name = family_graph.nodes[node]['name'].split()[0]
        header += f"{first_name:<12}"
    print(header)
    print("-" * (15 + 12 * len(family_graph.nodes())))

    # Beregn og vis matrisen
    for node1 in family_graph.nodes():
        first_name1 = family_graph.nodes[node1]['name'].split()[0]
        row = f"{first_name1:<15}"
        for node2 in family_graph.nodes():
            if node1 == node2:
                similarity = 1.0  # Perfekt similaritet med seg selv
            else:
                person1 = {attr: family_graph.nodes[node1][attr] for attr in␣
↪["gender", "age", "location", "profession", "name"]}
                person2 = {attr: family_graph.nodes[node2][attr] for attr in␣
↪["gender", "age", "location", "profession", "name"]}
                similarity = similarity_func(person1, person2)

            row += f"{similarity:<12.2f}"
        print(row)

    print("")

# Kombinert similaritets-mål
print("  Kombinert similaritets-mål / Combined similarity measure:")
print("")

def combined_similarity(person1, person2, weights=None):
    """Kombinert similaritet med vekter"""
    if weights is None:
        weights = {"Kjønn": 0.2, "Alder": 0.3, "Bosted": 0.2, "Yrke": 0.1,␣
↪"Navnelikhet": 0.2}

    total_similarity = 0
    for measure_name, weight in weights.items():
        similarity_func = similarity_measures[measure_name]
        similarity = similarity_func(person1, person2)
        total_similarity += weight * similarity

    return total_similarity

# Vis kombinerte similariteter
```

```
print("  Kombinert similaritet (vektet gjennomsnitt):")
print("  Combined similarity (weighted average):")
print("")

# Lag header
header = f"{'Node':<15}"
for node in family_graph.nodes():
    first_name = family_graph.nodes[node]['name'].split()[0]
    header += f"{first_name:<12}"
print(header)
print("-" * (15 + 12 * len(family_graph.nodes())))

# Beregn og vis matrisen
for node1 in family_graph.nodes():
    first_name1 = family_graph.nodes[node1]['name'].split()[0]
    row = f"{first_name1:<15}"
    for node2 in family_graph.nodes():
        if node1 == node2:
            similarity = 1.0
        else:
            person1 = {attr: family_graph.nodes[node1][attr] for attr in␣
 ↪["gender", "age", "location", "profession", "name"]}
            person2 = {attr: family_graph.nodes[node2][attr] for attr in␣
 ↪["gender", "age", "location", "profession", "name"]}
            similarity = combined_similarity(person1, person2)

        row += f"{similarity:<12.2f}"
    print(row)

print("")
print("="*60)
print("  FORKLARING AV SIMILARITETS-MÅL / EXPLANATION OF SIMILARITY MEASURES")
print("="*60)
print("  Norsk:")
print("• Kjønn: 1 hvis samme kjønn, 0 hvis forskjellig")
print("• Alder: Jo nærmere alder, jo høyere similaritet (0-1 skala)")
print("• Bosted: 1 hvis samme sted, 0 hvis forskjellig")
print("• Yrke: 1 hvis samme yrke, 0 hvis forskjellig")
print("• Navnelikhet: 1 hvis samme etternavn, 0 hvis forskjellig")
print("• Kombinert: Vektet gjennomsnitt av alle mål")
print("")
print("  English:")
print("• Gender: 1 if same gender, 0 if different")
print("• Age: Closer age = higher similarity (0-1 scale)")
print("• Location: 1 if same place, 0 if different")
print("• Profession: 1 if same profession, 0 if different")
print("• Name similarity: 1 if same surname, 0 if different")
```

```
print("• Combined: Weighted average of all measures")
print("")

# Vis interessante observasjoner
print("  Interessante observasjoner / Interesting observations:")
print("")
print("  Norsk:")
print("• Erik og Arvid: Høy similaritet (samme kjønn, etternavn, nær alder)")
print("• Ingrid og Anna: Høy similaritet (samme kjønn, nær alder)")
print("• Lars og Kari: Høy similaritet (samme bosted, yrke, etternavn)")
print("• Erik og Ingrid: Lav similaritet (forskjellig kjønn, yrke)")
print("")
print("  English:")
print("• Erik and Arvid: High similarity (same gender, surname, close age)")
print("• Ingrid and Anna: High similarity (same gender, close age)")
print("• Lars and Kari: High similarity (same location, profession, surname)")
print("• Erik and Ingrid: Low similarity (different gender, profession)")
print("")

# Fiks spring layout for konsistent visualisering
print("  Visualisering med fiksert layout / Visualization with fixed layout:")
print("")

# Definer fiksert posisjonering
fixed_positions = {
    "Bestefar": (0, 2),      # Øverst i midten
    "Bestemor": (1, 2),      # Øverst til høyre
    "Far": (0, 1),           # Midten til venstre
    "Mor": (1, 1),           # Midten til høyre
    "Barn1": (0, 0),         # Nederst til venstre
    "Barn2": (1, 0)          # Nederst til høyre
}

# Visualiser med fiksert layout
plt.figure(figsize=(16, 10))

# Separer noder etter kjønn
male_nodes = [node for node in family_graph.nodes() if family_graph.
 ↪nodes[node]['gender'] == 'male']
female_nodes = [node for node in family_graph.nodes() if family_graph.
 ↪nodes[node]['gender'] == 'female']

# Beregn node-størrelser basert på alder (proporsjonal)
ages = [family_graph.nodes[node]['age'] for node in family_graph.nodes()]
min_age, max_age = min(ages), max(ages)
age_range = max_age - min_age if max_age - min_age > 0 else 1
```

```
# Minimum og maksimum node-størrelse
min_size, max_size = 800, 3000

# Tegn noder med farger basert på kjønn og størrelse basert på alder
for node in male_nodes:
    age = family_graph.nodes[node]['age']
    # Normaliser alder til størrelse (eldre = større)
    normalized_age = (age - min_age) / age_range if age_range > 0 else 0.5
    node_size = min_size + (max_size - min_size) * normalized_age

    nx.draw_networkx_nodes(family_graph, fixed_positions, nodelist=[node],
                           node_color='blue', node_size=node_size, alpha=0.7)

for node in female_nodes:
    age = family_graph.nodes[node]['age']
    # Normaliser alder til størrelse (eldre = større)
    normalized_age = (age - min_age) / age_range if age_range > 0 else 0.5
    node_size = min_size + (max_size - min_size) * normalized_age

    nx.draw_networkx_nodes(family_graph, fixed_positions, nodelist=[node],
                           node_color='red', node_size=node_size, alpha=0.7)

# Tegn kanter med forskjellige farger
parent_child_edges = [(u, v) for u, v, d in family_graph.edges(data=True)
                      if d.get('relation') == 'parent-child']
marriage_edges = [(u, v) for u, v, d in family_graph.edges(data=True)
                  if d.get('relation') == 'marriage']

nx.draw_networkx_edges(family_graph, fixed_positions,␣
 ↪edgelist=parent_child_edges,
                       edge_color='black', arrows=True, arrowsize=20, width=2)
nx.draw_networkx_edges(family_graph, fixed_positions, edgelist=marriage_edges,
                       edge_color='gray', arrows=False, width=3, style='dashed')

# Legg til etiketter med fornavn
labels = {node: family_graph.nodes[node]['name'].split()[0] for node in␣
 ↪family_graph.nodes()}
nx.draw_networkx_labels(family_graph, fixed_positions, labels, font_size=12,␣
 ↪font_weight='bold')

# Legg til node-attributter som tekstfelt
for node, (x, y) in fixed_positions.items():
    attrs = family_graph.nodes[node]
    # Opprett tekst med attributter
    attr_text = f"Kjønn: {attrs['gender']}\nAlder: {attrs['age']} år\nBosted:␣
 ↪{attrs['location']}\nYrke: {attrs['profession']}"
```

```
    # Plasser tekstfelt enda nærmere noden
    plt.text(x + 0.08, y, attr_text, fontsize=8,
             bbox=dict(boxstyle="round,pad=0.3", facecolor='white', alpha=0.8,␣
 ↪edgecolor='gray'),
             verticalalignment='center')

plt.title("Slektstre med attributter / Family tree with attributes",␣
 ↪fontsize=16, pad=20)

# Legg til legende utenfor grafen
legend_elements = [
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='blue',␣
 ↪markersize=15, label='Menn / Men'),
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='red',␣
 ↪markersize=15, label='Kvinner / Women'),
    plt.Line2D([0], [0], color='black', linewidth=2, label='Forelder-barn /␣
 ↪Parent-child'),
    plt.Line2D([0], [0], color='gray', linewidth=3, linestyle='--',␣
 ↪label='Ekteskap/Partnerskap / Marriage/Partnership')
]

# Plasser legende lenger til høyre for å unngå overlapping
plt.legend(handles=legend_elements, loc='center left', bbox_to_anchor=(1.1, 0.
 ↪5), fontsize=12)

# Juster layout for å gi mer plass til legende
plt.tight_layout()
plt.subplots_adjust(right=0.65)

plt.axis('off')
plt.show()

print("")
print(" Visualiserings-forklaring / Visualization explanation:")
print(" Blå noder: Menn (størrelse proporsjonal med alder)")
print(" Røde noder: Kvinner (størrelse proporsjonal med alder)")
print(" Svarte piler: Forelder-barn relasjoner")
print(" Grå stiplede linjer: Ekteskap/Partnerskap")
print(" Hvite tekstfelt: Node-attributter (kjønn, alder, bosted, yrke)")
print("")
print(" Visualization explanation:")
print(" Blue nodes: Men (size proportional to age)")
print(" Red nodes: Women (size proportional to age)")
print(" Black arrows: Parent-child relationships")
print(" Gray dashed lines: Marriage/Partnership")
print(" White text boxes: Node attributes (gender, age, location, profession)")
```

[45]:
```
# Visualisering av alle similaritets-matriser / Visualization of all similarity␣
 ↪matrices
print(" Visualisering av similaritets-matriser / Visualization of similarity␣
 ↪matrices")
print("="*80)
print("")

import matplotlib.pyplot as plt
import numpy as np

# Sett opp figuren med subplots for alle matriser
fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('Similaritets-matriser / Similarity Matrices', fontsize=16,␣
 ↪fontweight='bold')

# Hent node-navn (fornavn)
node_names = [family_graph.nodes[node]['name'].split()[0] for node in␣
 ↪family_graph.nodes()]
nodes = list(family_graph.nodes())

# Definer similaritets-funksjoner (samme som tidligere)
def gender_similarity(person1, person2):
    return 1 if person1["gender"] == person2["gender"] else 0

def age_similarity(person1, person2):
    age_diff = abs(person1["age"] - person2["age"])
    max_age_diff = 60
    return max(0, 1 - (age_diff / max_age_diff))

def location_similarity(person1, person2):
    return 1 if person1["location"] == person2["location"] else 0

def profession_similarity(person1, person2):
    return 1 if person1["profession"] == person2["profession"] else 0

def name_similarity(person1, person2):
    name1_parts = person1["name"].split()
    name2_parts = person2["name"].split()
    return 1 if name1_parts[-1] == name2_parts[-1] else 0

def combined_similarity(person1, person2, weights=None):
    if weights is None:
        weights = {"Kjønn": 0.2, "Alder": 0.3, "Bosted": 0.2, "Yrke": 0.1,␣
 ↪"Navnelikhet": 0.2}

    similarity_measures = {
        "Kjønn": gender_similarity,
```

```
        "Alder": age_similarity,
        "Bosted": location_similarity,
        "Yrke": profession_similarity,
        "Navnelikhet": name_similarity
    }

    total_similarity = 0
    for measure_name, weight in weights.items():
        similarity_func = similarity_measures[measure_name]
        similarity = similarity_func(person1, person2)
        total_similarity += weight * similarity

    return total_similarity

# Lag similaritets-matriser
similarity_matrices = {}
similarity_names = ["Kjønn", "Alder", "Bosted", "Yrke", "Navnelikhet",␣
 ↪"Kombinert"]

for i, measure_name in enumerate(similarity_names):
    matrix = np.zeros((len(nodes), len(nodes)))

    for j, node1 in enumerate(nodes):
        for k, node2 in enumerate(nodes):
            if j == k:
                matrix[j, k] = 1.0  # Perfekt similaritet med seg selv
            else:
                person1 = {attr: family_graph.nodes[node1][attr] for attr in␣
 ↪["gender", "age", "location", "profession", "name"]}
                person2 = {attr: family_graph.nodes[node2][attr] for attr in␣
 ↪["gender", "age", "location", "profession", "name"]}

                if measure_name == "Kjønn":
                    matrix[j, k] = gender_similarity(person1, person2)
                elif measure_name == "Alder":
                    matrix[j, k] = age_similarity(person1, person2)
                elif measure_name == "Bosted":
                    matrix[j, k] = location_similarity(person1, person2)
                elif measure_name == "Yrke":
                    matrix[j, k] = profession_similarity(person1, person2)
                elif measure_name == "Navnelikhet":
                    matrix[j, k] = name_similarity(person1, person2)
                elif measure_name == "Kombinert":
                    matrix[j, k] = combined_similarity(person1, person2)

    similarity_matrices[measure_name] = matrix
```

```
# Visualiser hver matrise
positions = [(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
colors = ['Blues', 'Greens', 'Oranges', 'Purples', 'Reds', 'viridis']

for i, (measure_name, matrix) in enumerate(similarity_matrices.items()):
    row, col = positions[i]
    ax = axes[row, col]

    # Lag heatmap
    im = ax.imshow(matrix, cmap=colors[i], vmin=0, vmax=1)

    # Sett labels
    ax.set_xticks(range(len(node_names)))
    ax.set_yticks(range(len(node_names)))
    ax.set_xticklabels(node_names, rotation=45, ha='right')
    ax.set_yticklabels(node_names)

    # Legg til verdier i cellene
    for j in range(len(node_names)):
        for k in range(len(node_names)):
            text = ax.text(k, j, f'{matrix[j, k]:.2f}',
                           ha="center", va="center", color="black" if matrix[j,␣
 ↪k] < 0.5 else "white",
                           fontsize=8, fontweight='bold')

    # Sett tittel
    ax.set_title(f'{measure_name} / {measure_name}', fontweight='bold')

    # Legg til colorbar
    plt.colorbar(im, ax=ax, shrink=0.8)

plt.tight_layout()
plt.show()

# Analyser og sammenlign matrisene
print("  ANALYSE AV SIMILARITETS-MATRISER / ANALYSIS OF SIMILARITY MATRICES")
print("="*80)
print("")

# Finn høyeste og laveste similariteter
for measure_name, matrix in similarity_matrices.items():
    # Fjern diagonalen (selv-similaritet)
    mask = np.eye(matrix.shape[0], dtype=bool)
    off_diagonal = matrix[~mask]

    max_sim = np.max(off_diagonal)
    min_sim = np.min(off_diagonal)
```

```
    mean_sim = np.mean(off_diagonal)

    print(f"  {measure_name}:")
    print(f"    Høyeste similaritet: {max_sim:.3f}")
    print(f"    Laveste similaritet: {min_sim:.3f}")
    print(f"    Gjennomsnitt: {mean_sim:.3f}")

    # Finn par med høyeste similaritet
    max_indices = np.where(matrix == max_sim)
    for i, j in zip(max_indices[0], max_indices[1]):
        if i != j:  # Ikke diagonal
            print(f"    Høyeste: {node_names[i]}  {node_names[j]} ({max_sim:.
 ↪3f})")
    print("")

# Sammenlign matrisene
print("  SAMMENLIGNING AV MATRISER / MATRIX COMPARISON")
print("="*80)
print("")

# Korrelasjon mellom matrisene
correlation_matrix = np.zeros((len(similarity_names), len(similarity_names)))
for i, name1 in enumerate(similarity_names):
    for j, name2 in enumerate(similarity_names):
        if i == j:
            correlation_matrix[i, j] = 1.0
        else:
            # Flatten matrisene og beregn korrelasjon
            flat1 = similarity_matrices[name1].flatten()
            flat2 = similarity_matrices[name2].flatten()
            correlation = np.corrcoef(flat1, flat2)[0, 1]
            correlation_matrix[i, j] = correlation

# Visualiser korrelasjonsmatrisen
plt.figure(figsize=(10, 8))
im = plt.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1)

# Sett labels
plt.xticks(range(len(similarity_names)), similarity_names, rotation=45)
plt.yticks(range(len(similarity_names)), similarity_names)

# Legg til verdier i cellene
for i in range(len(similarity_names)):
    for j in range(len(similarity_names)):
        text = plt.text(j, i, f'{correlation_matrix[i, j]:.3f}',
                        ha="center", va="center",
```

```
                        color="white" if abs(correlation_matrix[i, j]) > 0.5␣
 ↪else "black",
                        fontsize=10, fontweight='bold')

plt.title('Korrelasjon mellom similaritets-matriser / Correlation between␣
 ↪similarity matrices')
plt.colorbar(im, label='Korrelasjon / Correlation')
plt.tight_layout()
plt.show()

# Identifiser interessante mønstre
print("  INTERESSANTE MØNSTRE / INTERESTING PATTERNS")
print("="*80)
print("")

# Finn par med høyest kombinert similaritet
combined_matrix = similarity_matrices["Kombinert"]
mask = np.eye(combined_matrix.shape[0], dtype=bool)
off_diagonal = combined_matrix[~mask]
max_combined = np.max(off_diagonal)

max_indices = np.where(combined_matrix == max_combined)
for i, j in zip(max_indices[0], max_indices[1]):
    if i != j:
        print(f"  Høyest kombinert similaritet: {node_names[i]}  ␣
 ↪{node_names[j]} ({max_combined:.3f})")

        # Analyser hvorfor de er like
        person1 = {attr: family_graph.nodes[nodes[i]][attr] for attr in␣
 ↪["gender", "age", "location", "profession", "name"]}
        person2 = {attr: family_graph.nodes[nodes[j]][attr] for attr in␣
 ↪["gender", "age", "location", "profession", "name"]}

        print(f"   Kjønn: {person1['gender']} vs {person2['gender']}␣
 ↪(similaritet: {gender_similarity(person1, person2):.1f})")
        print(f"   Alder: {person1['age']} vs {person2['age']} (similaritet:␣
 ↪{age_similarity(person1, person2):.3f})")
        print(f"   Bosted: {person1['location']} vs {person2['location']}␣
 ↪(similaritet: {location_similarity(person1, person2):.1f})")
        print(f"   Yrke: {person1['profession']} vs {person2['profession']}␣
 ↪(similaritet: {profession_similarity(person1, person2):.1f})")
        print(f"   Navn: {person1['name']} vs {person2['name']} (similaritet:␣
 ↪{name_similarity(person1, person2):.1f})")
        print("")

print("  Visualisering fullført!")
```

```
print("  Visualization completed!")
print("")
print("  INSIGHTS / INNSIKTER:")
print("• Heatmaps viser similaritets-mønstre visuelt")
print("• Korrelasjonsmatrise viser sammenhenger mellom attributter")
print("• Kombinert similaritet gir helhetsbilde av likhet")
print("• Farger indikerer styrke: mørk = høy similaritet, lys = lav␣
 ↪similaritet")
print("")
print("  INSIGHTS:")
print("• Heatmaps show similarity patterns visually")
print("• Correlation matrix shows relationships between attributes")
print("• Combined similarity gives holistic view of similarity")
print("• Colors indicate strength: dark = high similarity, light = low␣
 ↪similarity")
```

2.13 Del 4: Praktiske øvelser / Part 4: Practical Exercises

2.13.1 Refleksjonsoppgaver / Reflection Questions

1. Slektskap og grafer / Kinship and graphs - Hvilke typer grafer vil du bruke for å representere forskjellige slektsrelasjoner? - What types of graphs would you use to represent different family relationships?

2. Sykler i slektstrær / Cycles in family trees - Kan du tenke deg situasjoner hvor slektstrær kunne få sykler? - Can you think of situations where family trees could have cycles?

3. Kompleksitet / Complexity - Hvorfor blir slektstrær mer komplekse jo lenger tilbake i tid du går? - Why do family trees become more complex the further back in time you go?

2.13.2 Programmeringsoppgaver / Programming Exercises

Oppgave 1: Bygg et 3-personers slektstre / Exercise 1: Build a 3-person family tree

Bygg et enkelt slektstre med 3 personer og visualiser det.

Build a simple family tree with 3 people and visualize it.

[46]:
```
# LØSNING / SOLUTION
print("  Løsning til Oppgave 1 / Solution to Exercise 1")
print("  Solution to Exercise 1")

# Opprett et tomt slektstre
exercise_tree = nx.DiGraph()

# Legg til 3 personer med norske navn
exercise_tree.add_node("Forelder", name="Forelder Person", role="parent")
exercise_tree.add_node("Barn1", name="Første Barn", role="child")
exercise_tree.add_node("Barn2", name="Andre Barn", role="child")
```

```python
# Legg til relasjoner (kun forelder-barn relasjoner)
exercise_tree.add_edge("Forelder", "Barn1", relation="parent-child")
exercise_tree.add_edge("Forelder", "Barn2", relation="parent-child")
# Fjernet: exercise_tree.add_edge("Barn1", "Barn2", relation="siblings")
# Søsken er koblet gjennom foreldre, ikke direkte til hverandre

# Visualiser med fast seed for konsistent plassering
plt.figure(figsize=(10, 6))
# Bruk fast seed for konsistent node-plassering
pos = nx.spring_layout(exercise_tree, k=2, seed=42)

# Tegn noder med forskjellige farger basert på rolle
node_colors = ['lightgreen' if exercise_tree.nodes[node]['role'] == 'parent'
               else 'lightblue' for node in exercise_tree.nodes()]

nx.draw(exercise_tree, pos, with_labels=True, node_color=node_colors,
        node_size=2000, font_size=12, font_weight='bold',
        arrows=True, arrowsize=20)

plt.title("3-personers slektstre / 3-person family tree")
plt.show()

print(f"Antall personer: {exercise_tree.number_of_nodes()}")
print(f"Antall relasjoner: {exercise_tree.number_of_edges()}")
print(f"Number of people: {exercise_tree.number_of_nodes()}")
print(f"Number of relationships: {exercise_tree.number_of_edges()}")

# Vis slektsrelasjoner
print(f"\nSlektsrelasjoner:")
print(f"Family relationships:")
for u, v, data in exercise_tree.edges(data=True):
    print(f"  {exercise_tree.nodes[u]['name']} → {exercise_tree.
 ↪nodes[v]['name']} ({data['relation']})")

# Vis søsken-relasjon (implisitt)
print(f"\nSøsken-relasjon (implisitt):")
print(f"Sibling relationship (implicit):")
print(f"  {exercise_tree.nodes['Barn1']['name']} og {exercise_tree.
 ↪nodes['Barn2']['name']} er søsken")
print(f"  {exercise_tree.nodes['Barn1']['name']} and {exercise_tree.
 ↪nodes['Barn2']['name']} are siblings")
```

Oppgave 2: Finn stier mellom familiemedlemmer / Exercise 2: Find paths between family members

Bruk NetworkX til å finne alle mulige stier mellom to personer i et slektstre.

Use NetworkX to find all possible paths between two people in a family tree.

[47]:
```
# LØSNING / SOLUTION
print("  Løsning til Oppgave 2 / Solution to Exercise 2")
print("  Solution to Exercise 2")

def find_all_paths(graph, start, end, max_length=5):
    """Finn alle stier mellom to noder i en retnet graf."""
    try:
        # Bruk NetworkX for å finne alle enkle stier
        paths = list(nx.all_simple_paths(graph, start, end, cutoff=max_length))
        return paths
    except nx.NetworkXNoPath:
        return []

# Test med vårt familiegraf
start_person = "Bestefar"
end_person = "Barn1"

paths = find_all_paths(family_graph, start_person, end_person)

print(f"Stier fra {family_graph.nodes[start_person]['name']} til {family_graph.
 ↪nodes[end_person]['name']}:")
print(f"Paths from {family_graph.nodes[start_person]['name']} to {family_graph.
 ↪nodes[end_person]['name']}:")

for i, path in enumerate(paths, 1):
    path_names = [family_graph.nodes[node]['name'] for node in path]
    print(f"  {i}. {' → '.join(path_names)}")

if not paths:
    print("  Ingen stier funnet / No paths found")

# Finn også korteste sti
try:
    shortest_path = nx.shortest_path(family_graph, start_person, end_person)
    shortest_names = [family_graph.nodes[node]['name'] for node in␣
 ↪shortest_path]
    print(f"\nKorteste sti: {' → '.join(shortest_names)}")
    print(f"Shortest path: {' → '.join(shortest_names)}")
except nx.NetworkXNoPath:
    print("\nIngen sti funnet / No path found")
```

2.14 Del 5: Praktiske anvendelser / Part 5: Practical Applications

2.14.1 Hvordan slektstre-prosjektet bruker NetworkX / How the family tree project uses NetworkX

La oss se på hvordan det ekte slektstre-prosjektet bruker NetworkX:

Let's see how the real family tree project uses NetworkX:

[48]:
```
# Bruk datafilen som allerede er lastet
# Use the data file that is already loaded
print("Bruker allerede lastet familie_data")
print("Using already loaded familie_data")
```

2.15 Del 6: Oppsummering og neste steg / Part 6: Summary and Next Steps

2.15.1 Hva har du lært? / What have you learned?

I denne notebooken har du lært:

In this notebook you have learned:

1. **Slektstrær**: Hva de er, hvorfor de er viktige, og forskjellige typer
2. **Grafteori**: Grunnleggende konsepter som noder, kanter, og treer
3. **Kobling**: Hvordan slektstrær kan representeres som grafer
4. **Praksis**: Hvordan bruke NetworkX til å bygge og analysere slektstrær
5. **Øvelser**: Både refleksjon og programmering for å forstå begrepene

English: 1. **Family trees**: What they are, why they're important, and different types 2. **Graph theory**: Basic concepts like nodes, edges, and trees 3. **Connection**: How family trees can be represented as graphs 4. **Practice**: How to use NetworkX to build and analyze family trees 5. **Exercises**: Both reflection and programming to understand the concepts

2.15.2 Ordliste / Vocabulary

Norsk	English	Grafteori	Forklaring
Slektstre	Family tree	Tree	Spesiell type graf uten sykler
Person	Person	Node/Vertex	Et punkt i grafen
Slektsrelasjon	Family relationship	Edge	Linje mellom to noder
Forelder	Parent	-	Person som har barn
Barn	Child	-	Person som har foreldre
Generasjon	Generation	Distance	Avstand fra rot
Slektskap	Kinship	Path	Sti mellom to noder
Sykel	Cycle	Cycle	Sti som går i sirkel
Sammenheng	Connectivity	Connected	Alle noder kan nås
Komponent	Component	Component	Sammenhengende del av graf

2.15.3 Neste steg / Next Steps

Nå som du har lært grunnleggende konsepter, kan du gå videre til:

Now that you've learned the basic concepts, you can move on to:

1. `01_introduksjon.ipynb` - Oversikt over slektstre-prosjektet
2. `02_bygg_tre_manuelt.ipynb` - Bygge slektstrær programmatisk
3. `03_importer_data.ipynb` - Import og eksport av data
4. `04_visualisering.ipynb` - Avanserte visualiseringer
5. `05_eksterne_databaser.ipynb` - Integrasjon med eksterne databaser

2.15.4 Videre lesing / Further Reading

Grafteori / Graph Theory: - NetworkX Documentation - Introduction to Graph Theory - Graph Theory Tutorial

Genealogi / Genealogy: - FamilySearch - Verdens største genealogi-database - Digitalarkivet - Norske historiske kilder - Ancestry.com - Kommersiell genealogi-tjeneste

2.15.5 Takk for at du leste! / Thank you for reading!

Vi håper denne introduksjonen har gitt deg en god forståelse av både slektstrær og grafteori, og hvordan de kobles sammen i dette prosjektet.

We hope this introduction has given you a good understanding of both family trees and graph theory, and how they are connected in this project.

Lykke til med ditt eget slektstre-prosjekt!

Good luck with your own family tree project!

Denne notebooken er en del av slektstre-prosjektet. Se README.md for mer informasjon.

This notebook is part of the family tree project. See README.md for more information.

[ ]:
```yaml
# Last eksempel-familie data direkte i notebooken
import yaml
from io import StringIO

# Datafilen som tekst
data_text = """metadata:
  versjon: "1.0"
  opprettet: "2025-01-27T10:00:00"
  sist_endret: "2025-01-27T10:00:00"
  beskrivelse: "Eksempel familie med 4 generasjoner - Lundervold familien"

personer:
  # Generasjon 1 (eldste)
  - id: "p1"
    fornavn: "Erik"
```

```yaml
  etternavn: "Lundervold"
  kjønn: "male"
  fødselsdato: "1920-03-15"
  dødsdato: "1995-08-22"
  fødested: "Bergen"
  dødssted: "Oslo"
  notater: "Arbeidet som ingeniør på NSB"
  historier:
    - "Flyktet fra Norge under krigen"
    - "Bygde sitt eget hus i Oslo"

- id: "p2"
  fornavn: "Ingrid"
  mellomnavn: "Marie"
  etternavn: "Hansen"
  kjønn: "female"
  fødselsdato: "1925-07-10"
  dødsdato: "2010-12-03"
  fødested: "Trondheim"
  dødssted: "Oslo"
  notater: "Lærer og mor til 4 barn"
  historier:
    - "Møtte Erik på dans i 1947"
    - "Spilte piano og sang i kirkekor"

# Generasjon 2
- id: "p3"
  fornavn: "Arvid"
  etternavn: "Lundervold"
  kjønn: "male"
  fødselsdato: "1950-05-20"
  dødsdato: "2022-11-15"
  fødested: "Oslo"
  dødssted: "Bergen"
  notater: "Professor i informatikk"
  historier:
    - "Doktorgrad fra MIT"
    - "Grunnla flere teknologiselskaper"
  foreldre: ["p1", "p2"]

- id: "p4"
  fornavn: "Helena"
  mellomnavn: "Sofia"
  etternavn: "Lundervold"
  kjønn: "female"
  fødselsdato: "1952-09-12"
  fødested: "Oslo"
```

```yaml
  notater: "Arkitekt og kunstner"
  historier:
    - "Designet flere kjente bygninger i Bergen"
    - "Malte akvareller i fritiden"
  foreldre: ["p1", "p2"]

- id: "p5"
  fornavn: "Bjørn"
  etternavn: "Lundervold"
  kjønn: "male"
  fødselsdato: "1955-01-08"
  fødested: "Oslo"
  notater: "Lærer og fotballtrener"
  historier:
    - "Spilte fotball på høyt nivå i ungdommen"
    - "Trente juniorlag i 20 år"
  foreldre: ["p1", "p2"]

- id: "p6"
  fornavn: "Kari"
  etternavn: "Lundervold"
  kjønn: "female"
  fødselsdato: "1958-11-30"
  fødested: "Oslo"
  notater: "Sykepleier og mor til 3 barn"
  historier:
    - "Jobbet på sykehus i over 30 år"
    - "Var kjent for sin omsorgsfulle natur"
  foreldre: ["p1", "p2"]

# Generasjon 3
- id: "p8"
  fornavn: "Anna"
  mellomnavn: "Kristin"
  etternavn: "Pedersen"
  kjønn: "female"
  fødselsdato: "1952-08-25"
  fødested: "Stavanger"
  notater: "Markedsføringssjef"
  historier:
    - "Jobbet i flere internasjonale selskaper"
    - "Er aktiv i lokalsamfunnet"

- id: "p7"
  fornavn: "Magnus"
  etternavn: "Lundervold"
  kjønn: "male"
```

```
  fødselsdato: "1980-04-15"
  fødested: "Bergen"
  notater: "Dataingeniør og entreprenør"
  historier:
    - "Grunnla sitt første selskap som 22-åring"
    - "Reiser mye for jobb"
  foreldre: ["p3", "p8"]

- id: "p9"
  fornavn: "Erik"
  mellomnavn: "Arvid"
  etternavn: "Lundervold"
  kjønn: "male"
  fødselsdato: "1985-12-10"
  fødested: "Bergen"
  notater: "Forsker i kunstig intelligens"
  historier:
    - "Doktorgrad i maskinlæring"
    - "Publiserer regelmessig i internasjonale tidsskrifter"
  foreldre: ["p3", "p8"]

- id: "p10"
  fornavn: "Sofia"
  etternavn: "Lundervold"
  kjønn: "female"
  fødselsdato: "1988-06-18"
  fødested: "Bergen"
  notater: "Lærer og mor til 2 barn"
  historier:
    - "Jobber på samme skole som sin mor"
    - "Er aktiv i foreldreforeningen"
  foreldre: ["p3", "p8"]

- id: "p11"
  fornavn: "Lars"
  etternavn: "Andersen"
  kjønn: "male"
  fødselsdato: "1983-03-22"
  fødested: "Bergen"
  notater: "Musiker og komponist"
  historier:
    - "Spiller flere instrumenter"
    - "Har komponert musikk for filmer"
  foreldre: ["p4", "p17"]

- id: "p12"
  fornavn: "Marianne"
```

```
  etternavn: "Olsen"
  kjønn: "female"
  fødselsdato: "1985-09-14"
  fødested: "Bergen"
  notater: "Grafisk designer"
  historier:
    - "Jobber for et designbyrå"
    - "Har vunnet flere designpriser"
  foreldre: ["p4", "p17"]

- id: "p17"
  fornavn: "Ole"
  etternavn: "Andersen"
  kjønn: "male"
  fødselsdato: "1950-06-15"
  fødested: "Bergen"
  notater: "Ingeniør"
  historier:
    - "Jobbet i oljeindustrien"
    - "Var aktiv i lokalsamfunnet"

# Generasjon 4 (yngste)
- id: "p13"
  fornavn: "Emma"
  etternavn: "Lundervold"
  kjønn: "female"
  fødselsdato: "2010-02-28"
  fødested: "Bergen"
  notater: "Skoleelev, interessert i naturvitenskap"
  historier:
    - "Vinner av skolens vitenskapsmesse"
    - "Spiller fotball på lag"
  foreldre: ["p7", "p8"]

- id: "p14"
  fornavn: "Noah"
  etternavn: "Lundervold"
  kjønn: "male"
  fødselsdato: "2012-07-15"
  fødested: "Bergen"
  notater: "Skoleelev, interessert i teknologi"
  historier:
    - "Bygger roboter i fritiden"
    - "Er flink til matematikk"
  foreldre: ["p7", "p8"]

- id: "p15"
```

```yaml
    fornavn: "Maja"
    etternavn: "Lundervold"
    kjønn: "female"
    fødselsdato: "2015-11-03"
    fødested: "Bergen"
    notater: "Skoleelev, interessert i kunst og musikk"
    historier:
      - "Spiller piano og synger"
      - "Malte sitt første maleri som 6-åring"
    foreldre: ["p9", "p10"]

  - id: "p16"
    fornavn: "Oliver"
    etternavn: "Lundervold"
    kjønn: "male"
    fødselsdato: "2018-04-20"
    fødested: "Bergen"
    notater: "Skoleelev, interessert i dyr og natur"
    historier:
      - "Har en samling av steiner"
      - "Hjelper til på dyrehagen"
    foreldre: ["p9", "p10"]

ekteskap:
  - id: "e1"
    partner1_id: "p1"
    partner2_id: "p2"
    ekteskapsdato: "1947-06-15"
    ekteskapssted: "Oslo"
    ekteskapstype: "ekteskap"
    notater: "Gift i Oslo domkirke"

  - id: "e2"
    partner1_id: "p3"
    partner2_id: "p8"
    ekteskapsdato: "1978-08-20"
    ekteskapssted: "Bergen"
    ekteskapstype: "ekteskap"
    notater: "Gift i Bergen domkirke"

  - id: "e3"
    partner1_id: "p4"
    partner2_id: "p17"
    ekteskapsdato: "1980-05-10"
    ekteskapssted: "Bergen"
    ekteskapstype: "ekteskap"
    skilsmisse_dato: "1995-03-15"
```

```
    notater: "Skilt etter 15 års ekteskap"

  - id: "e4"
    partner1_id: "p7"
    partner2_id: "p8"
    ekteskapsdato: "2005-09-12"
    ekteskapssted: "Bergen"
    ekteskapstype: "ekteskap"
    notater: "Gift i Bergen rådhus"

  - id: "e5"
    partner1_id: "p9"
    partner2_id: "p10"
    ekteskapsdato: "2010-06-18"
    ekteskapssted: "Bergen"
    ekteskapstype: "ekteskap"
    notater: "Gift i Fantoft stavkirke"
"""

# Last data fra tekst
familie_data = yaml.safe_load(StringIO(data_text))

print(f"Familie lastet med {len(familie_data["personer"])} personer og␣
  ↪{len(familie_data["ekteskap"])} ekteskap")
print(f"Family loaded with {len(familie_data["personer"])} people and␣
  ↪{len(familie_data["ekteskap"])} marriages")
```