❓GREEDY SPANNER ALGORITHM
==========
The paper (Althöfer et al., 1993) introduced the greedy spanner algorithm, a foundational method for constructing a subgraph (spanner) that preserves the original graph's distances up to a small factor (stretch factor), while using a minimum number of edges and total edge weight. The work has been widely cited for its theoretical guarantees and practical applications in network design.

==========

✅ 1. THIS CODE IMPLEMENT A TRUE GREEDY SPANNER?

The greedy spanner algorithm is defined as:

For each edge (u, v) in non-decreasing weight order, add the edge only if:

d_H(u, v) > t · w

This ensures the spanner preserves distances up to stretch t while using fewer edges.

Our code checks:

```
dist = nx.shortest_path_length(H, u, v, weight='weight')
if dist > t * w:
    H.add_edge(u, v, weight=w)
```

✔️ This EXACTLY matches the greedy spanner definition.
✔️ For weight-1 edges and t=2, condition becomes:

dist > 2

—---- So,

✅ 1. DOES THIS CODE IMPLEMENT A TRUE GREEDY SPANNER?

Yes.
The implemented rule:

```
dist = nx.shortest_path_length(H, u, v, weight='weight')
if dist > t * w:
    H.add_edge(u, v)
```

matches the theoretical condition exactly.

With t = 2 and w = 1, the acceptance rule becomes:

Add (u, v) iff dist > 2

—--

✅ 2. HOW DOES IT WORK CORRECTLY FOR A COMPLETE GRAPH K_8?

Since all 28 edges have weight 1:

edges_sorted = lexicographically sorted edges

The greedy spanner on an unweighted complete graph is known to select:

✔️ A spanning tree
(7 edges)

And reject the remaining 21 edges because the path distance never exceeds 2.
—--

All edge weights = 1, so all edges tie in weight and are processed lexicographically:

(0,1), (0,2), (0,3), …, (0,7), (1,2), …

Initially, all nodes except 0 are disconnected, so for edges (0, x):

dist = ∞ → always > 2 → accept

Thus the algorithm accepts:

(0,1)
(0,2)
(0,3)
(0,4)
(0,5)
(0,6)
(0,7)

After these 7 edges are added, every pair of non-zero nodes has a 2-hop path through node 0:

i → 0 → j has length 2 ≤ 2

So every remaining edge is skipped.

✅ 3. FINAL OUTPUT

```
===== FINAL RESULT =====
Accepted edges (in order): [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7)]
Final edge count: 7
Final edges: [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7)]
========================================
```

The resulting spanner is a star centered at node 0, with degree 7 and total edges = 7.

This is a valid 2-spanner of K_8 and contains only O(n) edges.

```
===== BEGIN GREEDY SPANNER DEBUG LOG =====

STEP  1: ADD   edge=(0,1)   dist=inf   2*w=2   edges_now=1
STEP  2: ADD   edge=(0,2)   dist=inf   2*w=2   edges_now=2
STEP  3: ADD   edge=(0,3)   dist=inf   2*w=2   edges_now=3
STEP  4: ADD   edge=(0,4)   dist=inf   2*w=2   edges_now=4
STEP  5: ADD   edge=(0,5)   dist=inf   2*w=2   edges_now=5
STEP  6: ADD   edge=(0,6)   dist=inf   2*w=2   edges_now=6
STEP  7: ADD   edge=(0,7)   dist=inf   2*w=2   edges_now=7
STEP  8: SKIP  edge=(1,2)   dist=2   2*w=2   edges_now=7
STEP  9: SKIP  edge=(1,3)   dist=2   2*w=2   edges_now=7
STEP 10: SKIP  edge=(1,4)   dist=2   2*w=2   edges_now=7
STEP 11: SKIP  edge=(1,5)   dist=2   2*w=2   edges_now=7
STEP 12: SKIP  edge=(1,6)   dist=2   2*w=2   edges_now=7
STEP 13: SKIP  edge=(1,7)   dist=2   2*w=2   edges_now=7
STEP 14: SKIP  edge=(2,3)   dist=2   2*w=2   edges_now=7
STEP 15: SKIP  edge=(2,4)   dist=2   2*w=2   edges_now=7
STEP 16: SKIP  edge=(2,5)   dist=2   2*w=2   edges_now=7
STEP 17: SKIP  edge=(2,6)   dist=2   2*w=2   edges_now=7
STEP 18: SKIP  edge=(2,7)   dist=2   2*w=2   edges_now=7
STEP 19: SKIP  edge=(3,4)   dist=2   2*w=2   edges_now=7
STEP 20: SKIP  edge=(3,5)   dist=2   2*w=2   edges_now=7
STEP 21: SKIP  edge=(3,6)   dist=2   2*w=2   edges_now=7
STEP 22: SKIP  edge=(3,7)   dist=2   2*w=2   edges_now=7
STEP 23: SKIP  edge=(4,5)   dist=2   2*w=2   edges_now=7
STEP 24: SKIP  edge=(4,6)   dist=2   2*w=2   edges_now=7
STEP 25: SKIP  edge=(4,7)   dist=2   2*w=2   edges_now=7
STEP 26: SKIP  edge=(5,6)   dist=2   2*w=2   edges_now=7
STEP 27: SKIP  edge=(5,7)   dist=2   2*w=2   edges_now=7
STEP 28: SKIP  edge=(6,7)   dist=2   2*w=2   edges_now=7

===== FINAL RESULT =====
Accepted edges (in order): [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7)]
Final edge count: 7
Final edges: [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7)]
========================================
```

✔️ Theocratically:
Greedy(t=2) on complete graph → builds a tree → O(n) edges.

✅ 4. STATE SEQUENCE (28 STATES)

Since K_8 has 28 edges:

Total states recorded = 28

Steps 1–7: all (0, x) edges have dist = ∞ → add
Steps 8–28: every later edge has dist = 2 → skip

—-
✅ 4. CALCULATING THE 28-STEP STATE SEQUENCE

Since the edges are sorted lexicographically:

(0,1), (0,2), (0,3), ..., (0,7),
(1,2), (1,3), ..., (6,7)

For a complete graph of 8 nodes:

|V| = 8
|E| = 28

A new state is saved after each edge is processed, so:

Total states = 28

📌

Legend:

Add = condition satisfied (dist > 2)
Skip = edge rejected (dist <= 2)

✔ Greedy spanner ALWAYS forms a TREE here.

So it picks exactly 7 edges:

(0,1)
(1,2)
(2,3)
(3,4)

(4,5)
(5,6)
(6,7)

Every other edge gives path ≤ 2, so it is skipped.

✅ 28 STATES (H_1 to H_{28})

Each step corresponds to one iteration of the loop.

STEP 1: add (0,1) → edges = 1
STEP 2: add (0,2) → edges = 2 (path 0-2 = ∞)
STEP 3: skip (0,3)
STEP 4: skip (0,4)
STEP 5: skip (0,5)
STEP 6: skip (0,6)
STEP 7: skip (0,7)

STEP 8: add (1,2) → edges = 3
STEP 9: skip (1,3)
STEP 10: skip (1,4)
STEP 11: skip (1,5)
STEP 12: skip (1,6)
STEP 13: skip (1,7)

STEP 14: add (2,3) → edges = 4
STEP 15: skip (2,4)
STEP 16: skip (2,5)
STEP 17: skip (2,6)
STEP 18: skip (2,7)

STEP 19: add (3,4) → edges = 5
STEP 20: skip (3,5)
STEP 21: skip (3,6)
STEP 22: skip (3,7)

STEP 23: add (4,5) → edges = 6
STEP 24: skip (4,6)
STEP 25: skip (4,7)

STEP 26: add (5,6) → edges = 7
STEP 27: skip (5,7)

STEP 28: skip (6,7)

❗ FINAL VERIFIED RESULT

Total states: 28

Spanner edges: 7

Reduction: 75%

Final spanner:
A star centered at node 0,

0 connected to 1, 2, 3, 4, 5, 6, 7

Code ❓

```
======
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from IPython.display import HTML, display
import math

# ============================================================
# 1. Greedy Spanner WITH Debugging (prints each accepted edge)
# ============================================================

def greedy_spanner_with_states_debug(G, t=2):
    H = nx.Graph()
    H.add_nodes_from(G.nodes())

    # Deterministic sorting: first by weight, then by u, then v
    edges_sorted = sorted(G.edges(data=True),
                          key=lambda x: (x[2]['weight'], x[0], x[1]))

    states = []
    accepted_edges = []

    print("\n===== BEGIN GREEDY SPANNER DEBUG LOG =====\n")

    for step, (u, v, data) in enumerate(edges_sorted, start=1):
        w = data['weight']

        # shortest path distance in current H
        try:
            dist = nx.shortest_path_length(H, u, v, weight='weight')
        except nx.NetworkXNoPath:
```

```
            dist = math.inf

        # Rule: add only if dist > t * w
        if dist > t * w:
            H.add_edge(u, v, weight=w)
            accepted_edges.append((u, v))
            action = "ADD "
        else:
            action = "SKIP"

        print(f"STEP {step:2d}: {action}  edge=({u},{v})   dist={dist}   2*w={t*w}
edges_now={H.number_of_edges()}")

        # Save current state
        states.append(H.copy())

    print("\n===== FINAL RESULT =====")
    print("Accepted edges (in order):", accepted_edges)
    print("Final edge count:", H.number_of_edges())
    print("Final edges:", sorted(H.edges()))
    print("========================================\n")

    return H, states


# ============================================
# 2. Build Complete Graph K8 and Run Spanner
# ============================================

G = nx.complete_graph(8)

# Give all edges weight = 1
for u, v in G.edges():
    G[u][v]['weight'] = 1

# Run greedy spanner
H, states = greedy_spanner_with_states_debug(G, t=2)


# ============================================
# 3. Node positions for visualization
# ============================================

pos = nx.spring_layout(G, seed=42)


# ============================================
# 4. Animation
```

```
# ============================================

fig, ax = plt.subplots(figsize=(6, 6))

def draw_frame(i):
    ax.clear()
    ax.set_title(f"Greedy Spanner Construction – Step {i+1}", fontsize=13)

    # Draw full graph (gray)
    nx.draw_networkx_nodes(G, pos, node_color="lightgray", ax=ax)
    nx.draw_networkx_labels(G, pos, ax=ax)
    nx.draw_networkx_edges(G, pos, edge_color="lightgray", ax=ax)

    # Draw current spanner edges (blue)
    current_edges = states[i].edges()
    nx.draw_networkx_edges(
        G, pos,
        edgelist=list(current_edges),
        width=3,
        edge_color="blue",
        ax=ax
    )

    ax.text(
        0.5, -0.08,
        f"Blue edges = spanner edges at step {i+1} (count={states[i].number_of_edges()})",
        transform=ax.transAxes,
        ha='center',
        fontsize=10
    )

ani = FuncAnimation(fig, draw_frame, frames=len(states), interval=900, repeat=False)

display(HTML(ani.to_jshtml()))


# ============================================
# 5. Final Statistics
# ============================================

print("📊 GREEDY SPANNER STATISTICS")
print("----------------------------")
print("Original edges:", G.number_of_edges())
print("Spanner edges:", H.number_of_edges())
print("Reduction:", 100 * (1 - H.number_of_edges() / G.number_of_edges()), "%")
print("Connected:", nx.is_connected(H))
```

```
# OPTIONAL: Save animation
from matplotlib.animation import PillowWriter

ani.save("greedy_spanner_K8.mp4", writer="ffmpeg", fps=1)
ani.save("greedy_spanner_K8.gif", writer=PillowWriter(fps=1))
print("\n🎬 Animation saved as MP4 and GIF.")
```

```
======
```