PUNTO 1 G	“Rete di Brand concorrenti”
Vertici	**Brand** (brands.brand_id) venduti dallo **Store** selezionato
Archi (cond.)	Due brand **$B_1 \rightarrow B_2$** se nello stesso **mese** lo store ha venduto > *T* pezzi del brand B_1 **in più** di quelli del brand B_2 *(T fissato dall’utente)*
Peso	Somma, su tutti i mesi, del “delta positivo” di pezzi venduti
Input	Store, soglia **T** pezzi
Domande	• Crea grafo • Brand con **PageRank** massimo • Mese con densità di archi massima
PUNTO 2 G	“Scalata di prezzo”
Tema	**Scalata di prezzo fra brand** partendo dal brand con PageRank massimo
Vincoli ricorsivi	• Cammino semplice • Ad ogni passo si può passare solo a brand con **avg(list_price) più alto** • max L passi (utente)
Obiettivo	Massimizzare la **variazione di prezzo totale** (somma delle differenze di prezzo fra brand consecutivi)

◇ PUNTO 1 G – “Rete di Brand concorrenti”

1. Costruzione delle query SQL

1. Vertici
– Vogliamo tutti i brand_id che lo store ha venduto almeno una volta.

```
SELECT DISTINCT p.brand_id
FROM products p,
    order_items oi,
    orders o
WHERE p.product_id = oi.product_id
  AND oi.order_id   = o.order_id
  AND o.store_id    = %s;
```

Bind : (store_id,)

2. Vendite mensili per brand
– Per calcolare differenze mensili, aggrega vendite per brand e mese:

```
SELECT p.brand_id,
      MONTH(o.order_date) AS m,
```

```
      SUM(oi.quantity)    AS qty
FROM products p,
    order_items oi,
    orders o
WHERE p.product_id = oi.product_id
  AND oi.order_id   = o.order_id
  AND o.store_id    = %s
GROUP BY p.brand_id, MONTH(o.order_date);
```

Bind : (store_id,)

3. Archi e peso

– Fai un self-join sulla tabella precedente per generare tutti i $B_1 \to B_2$ che nel mese m hanno qty(B1)−qty(B2)>T\text{qty}(B_1)-\text{qty}(B_2) > T, e poi sommi tutti i delta positivi:

```
SELECT
  t1.brand_id AS b1,
  t2.brand_id AS b2,
  SUM(t1.qty - t2.qty) AS weight
FROM
  (
    /* vendite per brand, mese */
    …query precedente…
  ) t1,
  (
    /* stesse vendite per brand, mese */
    …query precedente…
  ) t2
WHERE t1.m = t2.m
  AND t1.brand_id <> t2.brand_id
  AND t1.qty - t2.qty > %s         -- soglia T
GROUP BY t1.brand_id, t2.brand_id;
```

Bind : (store_id, store_id, T)

2. Costruzione del grafo in Python

```
import networkx as nx

def build_brand_graph(store_id: int, T: int, dao) -> nx.DiGraph:
    G = nx.DiGraph()
    # 1) nodi
    brands = dao.get_brand_nodes(store_id)        # [brand_id, …]
    G.add_nodes_from(brands)
    # 2) archi
    for b1, b2, w in dao.get_brand_edges(store_id, T):
        G.add_edge(b1, b2, weight=w)
```

```
    return G
```

3. Risposte alle domande in Python

Brand con PageRank massimo

```
def brand_with_max_pagerank(G):
    pr = nx.pagerank(G, weight='weight')
    return max(pr, key=pr.get)
```

Mese con densità di archi massima
(ricicli l'aggregato mensile, ricostruisci il grafo mese-per-mese e calcoli densità = E/(V·(V–1)))

```
def month_max_edge_density(store_id, T, dao):
    best_m, best_d = None, -1
    for m in range(1,13):
        Gm = build_brand_graph_for_month(store_id, T, m, dao)
        V, E = Gm.number_of_nodes(), Gm.number_of_edges()
        if V>1:
            d = E / (V*(V-1))
            if d>best_d:
                best_d, best_m = d, m
    return best_m
```

◇ PUNTO 2 G – "Scalata di prezzo" fra brand

1. Preparazione

Calcola **avg_price[brand]** con una piccola query SQL:

```
SELECT brand_id, AVG(list_price) AS avg_price
FROM products
GROUP BY brand_id;
```

Dopo aver costruito il grafo di 1G, trova il **brand di partenza** con PageRank massimo:

```
start = brand_with_max_pagerank(G)
```

2. Backtracking in 3 funzioni Python

```
import copy

class PriceClimb:
    def __init__(self, G, avg_price: dict, L: int):
```

```
        self.G = G
        self.avg = avg_price
        self.L   = L
        self.best_path = []
        self.best_score = 0.0

    def getBest(self, start):
        # inizializza
        self.best_path = []
        self.best_score = 0.0
        # esplora ogni vicino con avg_price più alto
        for nbr in self.G.neighbors(start):
            if self.avg[nbr] > self.avg[start]:
                self._ricorsione([start, nbr])
        return self.best_path, self.best_score

    def _ricorsione(self, path):
        # aggiorna best se serve
        score = self.getScore(path)
        if score > self.best_score:
            self.best_score = score
            self.best_path = path.copy()
        # stop se supero L
        if len(path) >= self.L:
            return
        last = path[-1]
        for nbr in self.G.neighbors(last):
            if nbr not in path and self.avg[nbr] > self.avg[last]:
                path.append(nbr)
                self._ricorsione(path)
                path.pop()

    def getScore(self, path):
        # somma delle variazioni di prezzo
        return sum(self.avg[path[i+1]] - self.avg[path[i]]
                   for i in range(len(path)-1))
```

3. Utilizzo finale

```
# 1) costruisci grafo G con build_brand_graph(...)
# 2) calcola avg_price con dao.get_avg_price()
start = brand_with_max_pagerank(G)
climber = PriceClimb(G, avg_price_map, L)
best_path, best_delta = climber.getBest(start)
print("Scalata migliore:", best_path, "→ variazione:", best_delta)
```