#: 1C
Vertici: Clienti di uno store
Archi (condizione): Hanno acquistato lo stesso prodotto
Peso arco: # prodotti condivisi
Input utente: Store, prodotto facoltativo
Possibili domande:
• Cluster di clienti simili
• Cliente col maggior clustering-coefficient
#: 2C
Tema: Percorso tra store
Vincoli ricorsivi: Devono essere collegati da archi con #clienti crescente
Obiettivo da massimizzare: Lunghezza del percorso (n store)

1C – Grafo fra clienti di uno store

Query SQL (implicit joins)

1. Vertici: tutti i clienti che hanno fatto almeno un ordine nello store
(e, se passato, hanno comprato quel prodotto)

```
SELECT DISTINCT c.customer_id
FROM customers c,
    orders o,
    order_items oi
WHERE c.customer_id = o.customer_id
  AND o.order_id    = oi.order_id
  AND o.store_id    = %s
  AND oi.product_id = COALESCE(%s, oi.product_id);
```

Parametri: (store_id, product_id_or_NULL)

2. Archi non orientati: coppie di clienti A<B che hanno comprato insieme ≥1 prodotto
Peso = numero di prodotti distinti condivisi

```
SELECT
  oi1.customer_id   AS a,
  oi2.customer_id   AS b,
  COUNT(DISTINCT oi1.product_id) AS weight
FROM orders o1,
    order_items oi1,
    orders o2,
    order_items oi2
WHERE o1.order_id    = oi1.order_id
  AND o2.order_id    = oi2.order_id
  AND oi1.product_id = oi2.product_id
  AND o1.store_id    = %s
  AND oi1.product_id = COALESCE(%s, oi1.product_id)
```

```
  AND oi1.customer_id < oi2.customer_id
GROUP BY
  oi1.customer_id,
  oi2.customer_id;
```

Parametri: (store_id, product_id_or_NULL)

Analisi in Python: clustering coefficient

```
def compute_clustering_coefficient(graph):
    """
    Restituisce un dict {nodo: C(nodo)} dove
      C(v) = 2*#(archi tra vicini di v) / (deg(v)*(deg(v)-1))
    """
    coeffs = {}
    for v in graph.nodes():
        neigh = set(graph.neighbors(v))
        k = len(neigh)
        if k < 2:
            coeffs[v] = 0.0
            continue
        links = 0
        for u in neigh:
            for w in neigh:
                if u < w and graph.has_edge(u, w):
                    links += 1
        coeffs[v] = (2 * links) / (k * (k - 1))
    return coeffs

def find_max_clustering_coefficient(graph):
    """
    Restituisce la tupla (nodo, C) con C massimo.
    """
    coeffs = compute_clustering_coefficient(graph)
    return max(coeffs.items(), key=lambda kv: kv[1])
```

2C – Percorso fra store con archi “#clienti crescente”

```
import copy

class StorePathFinder:
    def __init__(self, graph):
        """
        :param graph: grafo NetworkX orientato, nodi = store,
                      ogni arco u→v ha attributo 'weight' = # clienti condivisi
```

```
        """
        self.graph = graph
        self.best_path = []
        self.best_length = 0

    def getBestPath(self, start):
        """
        Inizializza la ricerca e ritorna (best_path, best_length).
        start: nodo di partenza (store)
        """
        self.best_path = []
        self.best_length = 0
        for v in self.graph.neighbors(start):
            self._ricorsione([start, v])
        return self.best_path, self.best_length

    def _ricorsione(self, path):
        """
        path: lista corrente di store
        Applica backtracking vincolato a archi di peso crescente.
        """
        # Valuta lunghezza (numero di store)
        length = self.getScore(path)
        if length > self.best_length:
            self.best_length = length
            self.best_path = path.copy()

        prev, last = path[-2], path[-1]
        prev_w = self.graph[prev][last]['weight']
        for nbr in self.graph.neighbors(last):
            if nbr not in path:
                new_w = self.graph[last][nbr]['weight']
                # vincolo: peso crescente
                if new_w > prev_w:
                    path.append(nbr)
                    self._ricorsione(path)
                    path.pop()

    def getScore(self, path):
        """
        Punteggio = lunghezza del percorso (numero di nodi).
        """
        return len(path)
```

Come usarli

1C: esegui le due query, costruisci il grafo con nx.Graph(), chiama compute_clustering_coefficient(g) o find_ma›

2C: costruisci il grafo dei store (archi con weight = # clienti condivisi),
poi **restituendo la sequenza di store più lunga con archi "in crescita".**

```
finder = StorePathFinder(store_graph)
best_path, length = finder.getBestPath(start_store)
```

In questo modo hai **solo join impliciti** in SQL e **codice Python reale** per clustering e backtracking, pronto per esse

c_clustering_coefficient(g).

ere incollato nei tuoi moduli.