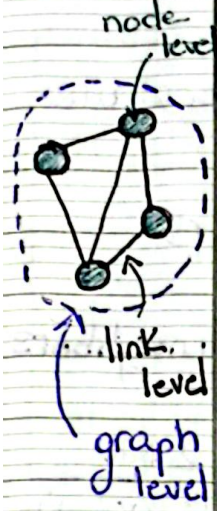
Machine Learning for Graph Predictions



→ Overview: Three Types of Graph Predictions

1) Node-Level
- Predict properties, labels of individual nodes.
- ↳ ex: Node classification (predicting if a user is influential)

2) Link-Level
- Predict new connections between nodes.
- ↳ ex: Friend recommendation

3) Graph-Level
- Predict properties of entire graphs
- ↳ ex: Molecular property prediction, graph classification

→ ML Pipeline for Graphs

1) Feature Design: Convert nodes/links/graphs into d-dimensional vectors
2) Model Design: Train models (e.g. Random Forest, SVM, Neural Networks, etc.)
3) Prediction: Use model to make predictions

→ Node-Level Features

- Goal: Characterize the structure and position of a node in the network

Importance-Based Features

1) Node Degree (k_v): Number of neighboring nodes
2) Node Centrality Measures (C_v)

↳ Eigenvector
- A node is important if surrounded by important neighbors
- Formula: $Ax = \lambda x$
- Centrality is the eigenvector (x).

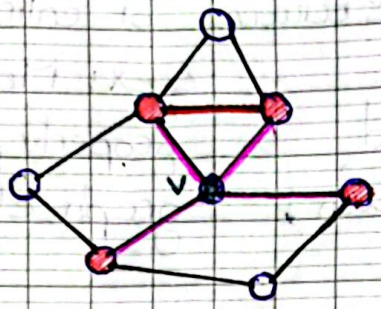
↳ Betweenness
- A node is important if it lies on many shortest path between other nodes
- Formula: $\sum \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}$
  - ↳ # shortest path between s,t containing v / # shortest path between s,t

↳ Closeness
- A node is important if it has short path to all other nodes
- Formula: $1/\sum$ (shortest path length from v to u)

Structure Based Features

1) Cluster Coefficient:

- Measures how connected a node's neighbors are
- Example:



v: a node
Kv: its degree
Nv: nb. of links between v neighbors

$N_v = 1$, $K_v = 4 \Rightarrow C(v) = \frac{2N_v}{K_v(K_v-1)} = \frac{2(1)}{4(4-1)} = \frac{1}{6}$

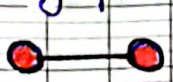
2) Graphlets

- Rooted, small (2-5 nodes), connected (all nodes are reachable from each other), non-isomorphic (represent a unique structural pattern) subgraphs

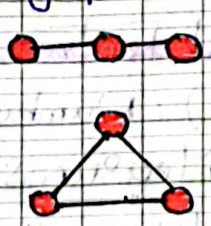
- Considering graphlets of size 2-5 nodes we get 73 different possible shapes (graphlets)

↳ Some graphlet examples

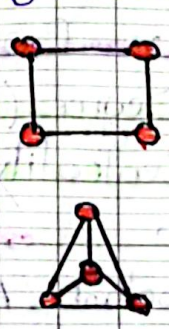
2-node graphlets



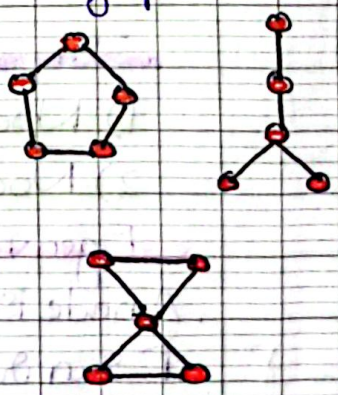
3node graphlets



4-node graplets



5node graphlets



↳ graphlet calculation

$n = 20$, $m = 180$

1) total possible nb. of edges (every vertex connected to every other one)

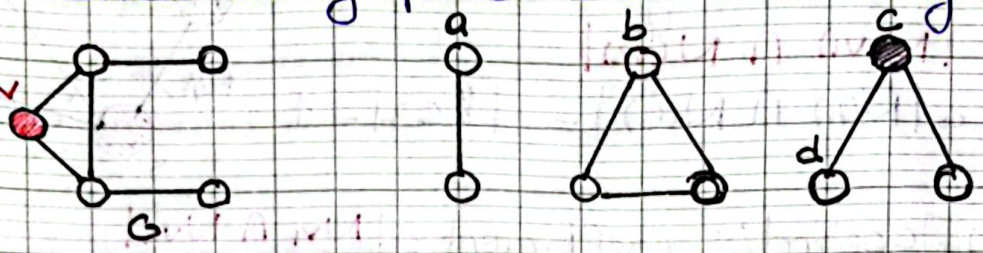
$K = n(n-1)/2 = (20 \cdot 19)/2 = 190$

2) nb. of all possible graphs: $S = C(n, K) = C(180, 190)$

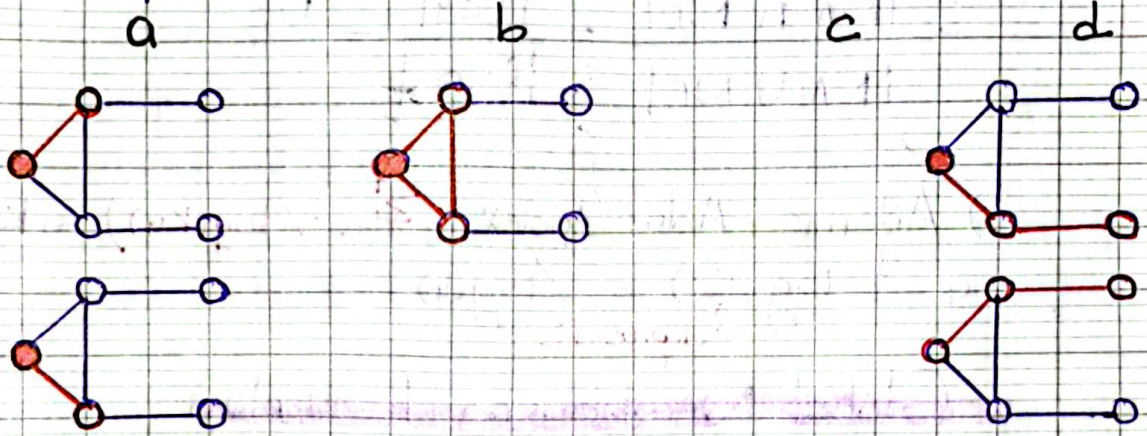
↳ Graphlet Degree Vector (GDV)

- Count vector of graphlets containing the target node
- Example:

Given the graph G and a list of graphlets



Graphlet instance



⇒ GDV of node v:

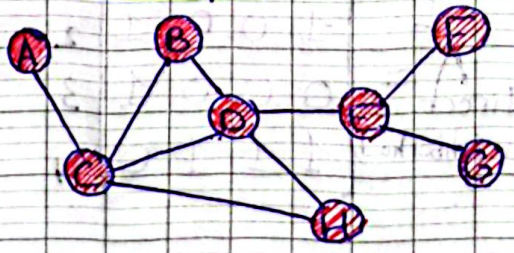
$[\overset{a}{2}, \overset{b}{1}, \overset{c}{0}, \overset{d}{2}]$

→ Link-Level Features

- Goal: predict new links based on existing links
  - ↳ The key is to design features for a pair of nodes

* Distance-Based Features
  - ↳ Shortest path distance between two nodes

Example:



$S_{BH} = S_{BE} = S_{AB} = 2$

$S_{BG} = S_{BF} = 3$

↳ limitation: doesn't capture neighborhood overlap

limitation: metric is 0 if the two nodes doesn't have any neighbors in common

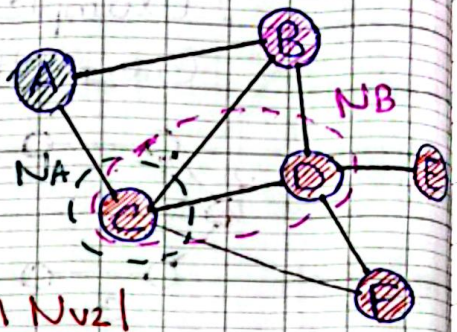
* Local Neighborhood Overlap

Captures nb. of neighboring nodes shared between two nodes v_1 and v_2

1) Common neighbors

$|N(v_1) \cap N(v_2)|$

↳ $|N(A) \cap N(B)| = |\{C\}| = 1$



2) Jaccard's coefficient : $\frac{|N_{v_1} \cap N_{v_2}|}{|N_{v_1} \cup N_{v_2}|}$

↳ $\frac{|N_A \cap N_B|}{|N_A \cup N_B|} = \frac{|\{C\}|}{|\{C,D\}|} = 1/2$

3) Adamic-Adar Index : $\sum (1/\log(k_u)) \Rightarrow u \in N_{v_1} \cap N_{v_2}$

↳ $1/\log(K_C) = 1/\log(4)$

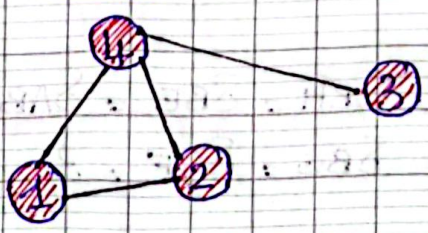
↑ degree of C

* Global Neighborhood Overlap

↳ resolve the local neighborhood overlap limilation, it can capture potential connections even without shared neighbors

↳ Katz Index : count the nb. of paths of all lengths between nodes

- let P^K_{uv} = # paths of length K between u and v (P^1_{uv} : # paths of length 1 (direct neighborhood) between u,v)
- we will show $P^K = A^K$
- ex:



$P^{(1)}_{12} = A_{12}$

A^1 (direct neighborhood) =

	1	2	3	4	
	0	1	0	1	1
	1	0	0	1	2
	0	0	0	1	3
	1	1	1	0	4

$\Rightarrow A^2$: specifies # path of length 2 between u and v

A^L : " " " " " L " " "

→ Graph_Level Features

• Kernel Methods

↳ Widely-used for Traditional ML for graph-level pred.

↳ Kernel Function : $K(G_1, G_2)$ measures similarity between graphs

↳ Kernel Matrix : $K = (K(G, G'))_{G,G'}$

1) Graphlet Kernel (Bag_of_graphlets)

. Count the nb. of different graphlets in a graph

. Note : Definition of graphlets here is slightly different from node_level features

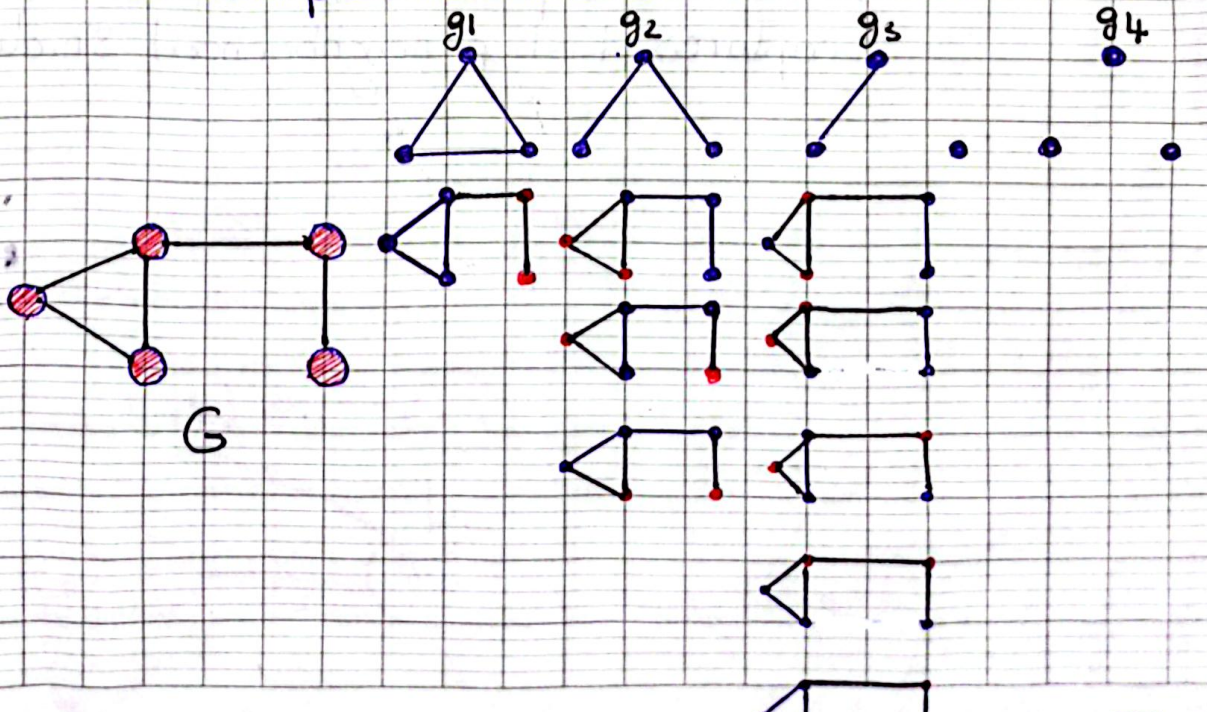
↳ Nodes in graphlets here don't need to be connected (we can have isolated nodes)

↳ Graphlets here are not rooted

. Graphlet Features

Given a graph G, and a graphlet list $G_k = (g_1, g_2, \dots, g_{n_k})$, define the graphlet count vector f_G as : $(f_G)_i = \#(g_i \subseteq G)$ For $i = 1, 2, \dots, n_k$.

↳ example : for $k = 3$



$f_G = (1, \quad 3, \quad 6, \quad 0)^T$

- Graphlet Kernel

  Given two graphs, G and G', graphlet Kernel is computed as: $K(G, G') = f_G^T f_{G'}$

  ↳ problem: if G and G' have different sizes, that will skew the value

  ↳ Solution: normalize each feature vector

  $h_G = F(G) / Sum(f_G)$, $K(G, G') = h_G^T h_{G'}$

  ↳ Counting graphlets is expensive

2) Weisfeiler_Lehman Kernel (Bag_of_colors)

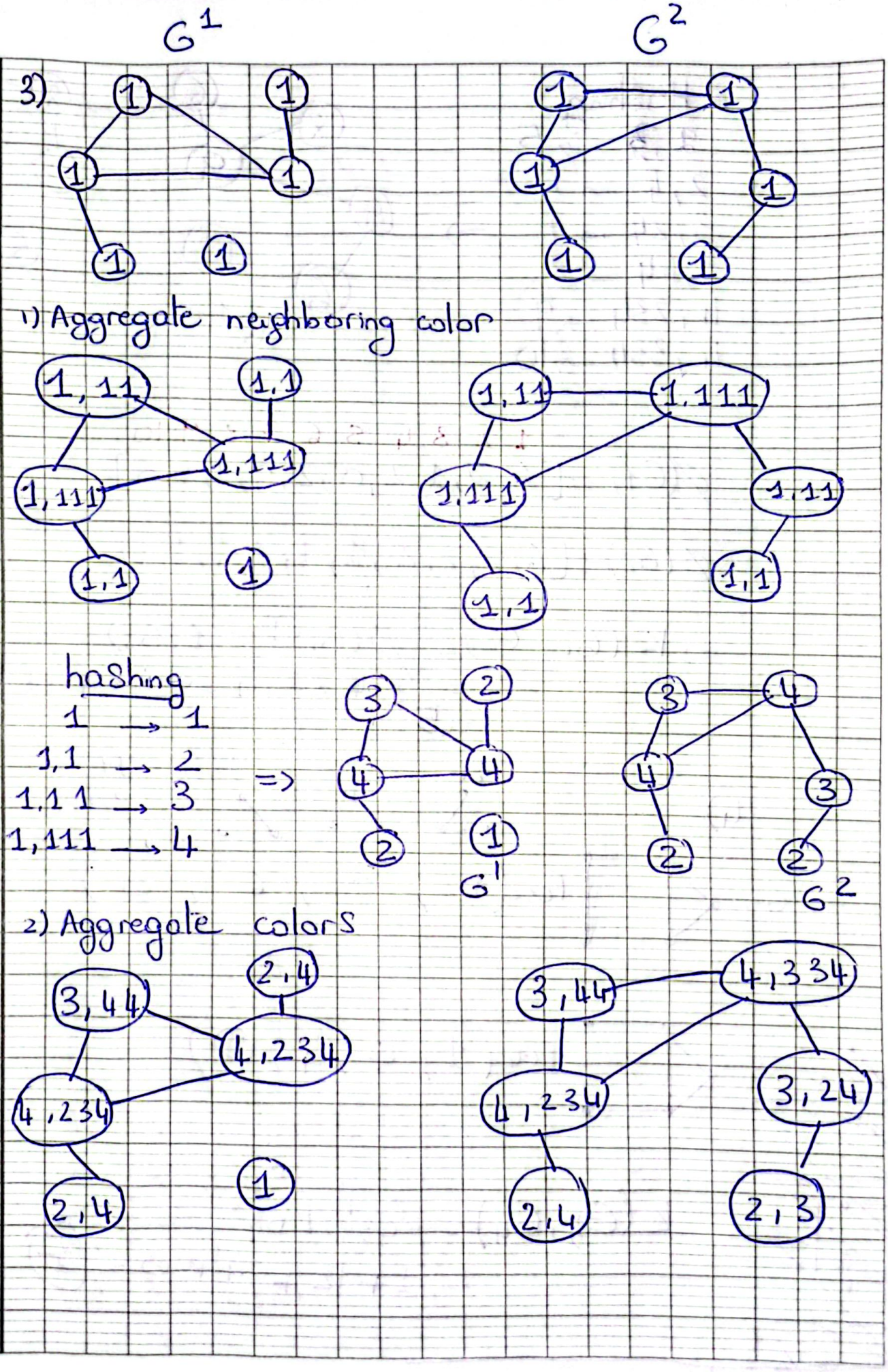
- Goal: Design an efficient graph feature
- Idea: Use neighborhood structure to iteratively enrich node vocabulary
- Algorithm: Color refinement

↳ Color Refinement

Given a graph G with a set of nodes V

1) Assign initial color $c^{(0)}v$ to each node
2) Iterate: $C_{(v)}^{(k+1)} = HASH(\{C_{(v)}^{k}, \{C^{(k)}(u)\}_{u \in N(v)}\})$
3) After K steps of color refinement, $c^{(K)}(v)$ captures K-hop neighborhood structure

-Weisfeiler - Lehman Kernel algorithm example



Hashing

2, 3 → 5

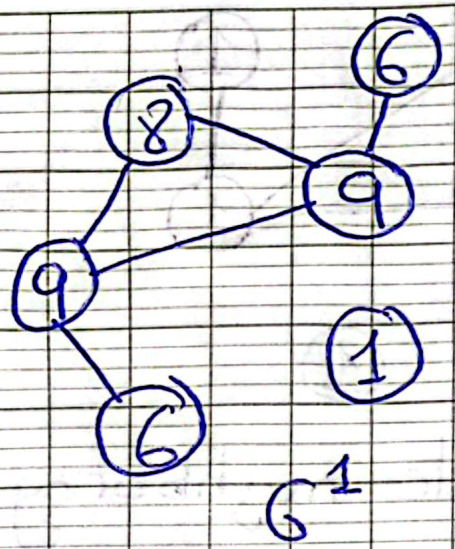
2, 4 → 6

3, 24 → 7

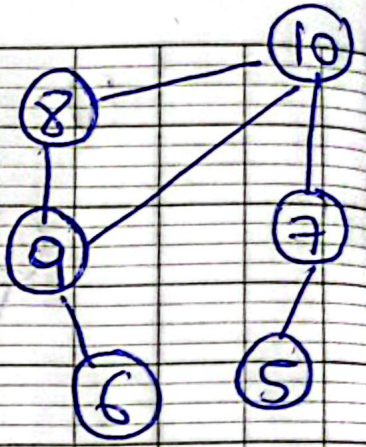
3, 44 → 8

4, 234 → 9

4, 334 → 10



G^1



G^2

	1	2	3	4	5	6	7	8	9	10
$\phi(G_1)$ =	6	2	1	2	0	2	0	1	2	0

$\phi(G_2) = [6, 2, 2, 2, 1, 1, 1, 1, 1, 1]$

$K(G_1, G_2) = \phi(G_1)^T \cdot \phi(G_2)$

$= 36 + 4 + 2 + 4 + 0 + 2 + 0 + 1 + 2 + 0$

$= 51$