Creating Partitions
Creating Partitions

- $V$ is partitioned into $P$ disjoint subsets $V_i$s of consecutive nodes, such that, for any $j$ and $k$, $V_j \cap V_k = \emptyset$ and $\bigcup_k V_k = V$
- $V_i' = V_i \cup \{v : v \in N_u, u \in V_i\}$
- $E_i' = \{(u, v) : u \in V_i, v \in N_u\}$
Creating Partitions

$$\sum_{v \in V_i} f(v) \approx \frac{1}{P} \sum_{v \in V} f(v).$$
Creating Partitions

\[ f(v) = \sum_{u \in \mathcal{N}_v - \mathcal{N}_v} (d_v + d_u), \]
Creating Partitions

```c
struct Vertex {
    struct Edge *edges1;
    struct Edge *edges2;
    int numNeighbors;
    int numOrderedNeighbors;
    int numOtherNeighbors;
    int *neighbors;
    int *orderedNeighbors;
    int *otherNeighbors; // neighbors - orderedNeighbors
};
```
void createNeighborSets(struct Graph *g) {
    int i, j, k, v;
    for (v = 0; v < g->numVertices; v++) {
        g->vertices[v].neighbors = (int *) malloc(sizeof(int)*g->vertices[v].numNeighbors);
        g->vertices[v].orderedNeighbors = (int *) malloc(sizeof(int)*
            g->vertices[v].numNeighbors);
        g->vertices[v].otherNeighbors = (int *) malloc(sizeof(int)*g->vertices[v].numNeighbors);
    }
}
Creating Partitions

for (v = 0; v < g->numVertices; v++) {
    struct Edge *e = g->vertices[v].edges1;
    i = 0; j = 0; k = 0;
    while (e != NULL) {
        g->vertices[v].neighbors[i] = e->vertex2;
        i++;
        if (degree(g, v) <= degree(g, e->vertex2)) {
            g->vertices[v].orderedNeighbors[j] = e->vertex2;
            j++;
        } else {
            g->vertices[v].otherNeighbors[k] = e->vertex2;
            k++;
        }
        e = e->next1;
    }
}
Creating Partitions

e = g->vertices[v].edges2;
    while (e != NULL) {
        g->vertices[v].neighbors[i] = e->vertex1;
        i++;
        if (degree(g, v) < degree(g, e->vertex1)) {
            g->vertices[v].orderedNeighbors[j] = e->vertex1;
            j++;
        } else {
            g->vertices[v].otherNeighbors[k] = e->vertex1;
            k++;
        }
    e = e->next2;
}
g->vertices[v].numNeighbors = i;
g->vertices[v].numOrderedNeighbors = j;
g->vertices[v].numOtherNeighbors = k;