The partitioning subpart of the project is the first step in your layout system design. In this assignment, you are asked to implement a partitioning algorithm based on a hybrid of the Kernighan-Lin and the Fiduccia-Mattheyses algorithms.

**Overview:**

*Optimization Goal:* The partitioning algorithm that you will be implementing aims at minimizing the interconnects crossing the two partitions. This will be subject to an area constraint, as you have learned in the course. The algorithm will start by first obtaining an initial partitioning. Although complex initial partitioning algorithms exist, for simplicity’s sake, we will go with a random partitioning for this part. The initial random partitioning will be iteratively improved by migrating cells between partitions. Your algorithm terminates when no further improvement is possible.

*Interconnect model:* A net in a circuit can connect multiple pins (one output pin and multiple input pins). Thus the accurate model for a net should be a hyperedge which is quite difficult to handle in an algorithm. For simplicity’s sake, you should model the interconnects between cells as edges in a full graph. Note that only the output-input connections need to be modeled. The input-input connections should be ignored as such connections are not the primary concern in layout optimization. Your algorithm should aim at minimizing the number of cut *edges*, using the gain equations in KL, therefore.

*Constraints:* As you have learned in the course, a practical design usually has area constraints for each partition and thus avoids highly skewed partitions. Consequently, you also need to impose a certain tolerance band in terms of the partition size in your algorithm. The tolerance band should be set to the maximum size of a component so as to enable the possible movement of all components. More specifically, you should use the following balance criterion: \( \text{floor}(|V|/2 - C_{\text{max}}) \leq |A| \leq \text{ceiling}( |V|/2 + C_{\text{max}}) \). Of course, be sure that your initial random partition meets the balance criterion.

**Algorithm:**

- Initially create two random partitions.
- Calculate initial gains of moving each component to the other partition using the KL gain equation.
- Choose the component with the highest gain, which satisfies the size constraint, and tentatively move it to the other partition. Lock this component so that it will not be considered during this iteration. Use an efficient algorithm for updating the gains.
- Repeat moving the components from one partition to another until no more components can be moved.
- Make permanent all the movements up to and including the move which gives the highest accumulative gain.

* The function \( \text{floor} \) truncates the fractional part of its argument, while the function \( \text{ceiling} \) rounds it up to the smallest integer that exceeds the argument. \(|V|\) is the total size of the cells. \(|A|\) denotes the size of a single partition. The number \( C_{\text{max}} \) represents the size of the largest cell.
Unlock all components and repeat until no further improvements can be achieved.

**Data structure design:**

By examining the algorithm, you may find that in each iteration you need to make as many tentative moves as the number of components and make a subset of them permanent. While the time complexity is linear, each of these macro move operations might be expensive as each consists of calculating the gain and choosing the component with highest gain. These low level operations may consume significant time and storage space when the number of components is large. As efficiency is one of the key criteria for evaluating an EDA tool, you should try to identify and use the most efficient data structure to implement the algorithm. A simple implementation of course would consist of always calling the initial gain calculation subroutine after every tentative move, and thus identifying the highest gain component. While this strategy certainly works, it is ineffective as the gain of some components (e.g. those that have already been locked) need not be updated and the search process may take a lot of time if the number of components is quite large. To ameliorate this unnecessary time expenditure, we propose to you three possible data structures. Analyze each of them in terms of time and space efficiency and choose the best one you think. Of course, don’t be solely content with our suggestions; if you have a better idea for the right data structure, make a case for it in your writeup and go for it.

**Data structure 1:**

Create a bucket array stamped by the possible gain values from –Pmax to Pmax*. Each element of the array points to a linked list within which each item is an unlocked component whose gain under the current partition is equal to the gain value stamped at the head of the linked list. Use a specific MaxGain index to keep track of the highest gain value under the current partition. Whenever the unlocked component group is non-empty, choose one component in the linked list pointed to by MaxGain for a tentative move by deleting it from the linked list, and updating the gain (position in the bucket) of the remaining unlocked components.

**Data structure 2:**

Create a gain matrix $G$ where each row corresponds to a specific gain value and each column to a component. Assign 1 to the corresponding entry, $G_{ij}$, if the gain of component $j$ is $i$ under the current partition. Use a specific MaxGain index to keep track of the highest gain value under the current partition. Whenever the unlocked component group is non-empty, choose one component in the row pointed to by MaxGain for a tentative move, and update the gain of the remaining unlocked components by updating the gain matrix.

**Data structure 3:**

Create a gain array where the value of element $i$ denotes the gain of component $i$ under the current partition. Whenever the unlocked component group is non-empty, choose the component with the maximum gain value for a tentative move, and update the gain of the remaining unlocked components by updating the gain array.

*Pmax is the maximum degree of the vertex in the graph modeling the circuit.*
One thing that should be noted is that none of the aforementioned data structures is by itself sufficient to completely support the algorithm. The gain update requires also examination of the connectivity matrix and of your current partition, which you of course will be updating after each move.

**Implementation hints:**
The randomness of the initial partitioning might increase the difficulty in your code debugging. If you are using the C language to implement the code, you can simply use the `rand()` function *without* initializing the seed. This will constantly generate the same sequence of pseudorandom numbers, which may help you analyze the behavior of your program by keeping the initial partition identical for each debugging run.

Although the suggested data structures might improve the efficiency of your program, their implementation imposes additional challenges over and above the simple strategy of always invoking the initial gain calculation subroutine. You can implement the simpler scheme as a reference and compare your real turn-in version of implementation for detecting any possible errors.

**Input Specification:**
The input format (which you will use for all your projects) is as the format specified in the overall project description, which is repeated in Figure 1 for your convenience. The first line in the file indicates the number of components, $N$, in the circuit. The following $N$ lines correspond to the connectivity matrix, $A$, of the circuit, and the last line indicates the sizes of the components in the circuit. The non-zero entries, $A_{ij}$, in matrix $A$ denote a connection between components $i$ and $j$ in the circuit. As we discussed in the overall project description document, for the connection between $i$ and $j$, $A_{ij}$ indicates the pin number of the component $i$, and $A_{ji}$ indicates the pin number of the component $j$. Yet in the first project, you need not use the pin number information as the partition problem only cares about the connectivity relationship between cells. You can abstract away the pin number information and only utilize the connectivity information that you need for this part, simply by checking for whether the number encountered in the matrix is (non)zero.

If you look at the example given, you will notice that there is a net connecting components 2, 5, and 6. The same holds for the cells 6, 7, and 8. Throughout this project we will consider only the edges from the net that connect output to input pins and only these edges will be given in the connectivity matrix.
Output Format:
After reading in the circuit information, your program should output the number of components, total circuit size, the maximum component size, and the initial partitions, and initial cutset size.

The number of components: 11
Total size of the components: 17
The maximum component size: 2
Partition 0: 0 1 2 3 4 5 6
Partition 1: 7 8 9 10
Initial cutset size: 5

Subsequently, for each iteration, the components selected to be moved should be displayed with the current total gain of the movements made so far.

Move 1 from Partition 0 to Partition 1. Gain: 1
Move 4 from Partition 0 to Partition 1. Gain: 2
Move 9 from Partition 1 to Partition 0. Gain: 2
Move 6 from Partition 0 to Partition 1. Gain: 2
Move 3 from Partition 0 to Partition 1. Gain: 3
Move 7 from Partition 1 to Partition 0. Gain: 1
Move 2 from Partition 0 to Partition 1. Gain: 1
Move 10 from Partition 1 to Partition 0. Gain: 1
Move 0 from Partition 0 to Partition 1. Gain: 0
Move 5 from Partition 0 to Partition 1. Gain: 1
Move 8 from Partition 1 to Partition 0. Gain: 0

At the end of each iteration, the maximum gain attained during the iteration, the final partitions and the final cutset size should be displayed.
Max Gain: 3
Partition 0:  0 9 2 5
Partition 1:  7 8 4 10 1 6 3
Final Cutset Size: 2

Move  4 from Partition 1 to Partition0.        Gain: -1
Move  2 from Partition 0 to Partition1.        Gain: -1
Move  8 from Partition 1 to Partition0.        Gain: -2
Move  9 from Partition 0 to Partition1.        Gain: -2
Move 10 from Partition 1 to Partition0.        Gain: -2
Move  5 from Partition 0 to Partition1.        Gain: -1
Move  3 from Partition 1 to Partition0.        Gain: -2
Move  0 from Partition 0 to Partition1.        Gain: -1
Move  6 from Partition 1 to Partition0.        Gain: -1
Move  7 from Partition 1 to Partition0.        Gain: -1
Move  1 from Partition 1 to Partition0.        Gain: 0
Max Gain: 0
Partition 0:  0 9 2 5
Partition 1:  7 8 4 10 1 6 3
Final Cutset Size: 2

Additional self-test examples:
The following two examples are provided to you for self-testing your program.

<table>
<thead>
<tr>
<th>Example 1</th>
<th>Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0</td>
<td>0 0 0 0 1 1 1 1</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0</td>
<td>0 0 0 1 0 1 1 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 1 1</td>
<td>0 0 0 1 1 0 1 0</td>
</tr>
<tr>
<td>0 0 0 0 0 1 1</td>
<td>0 1 0 0 0 0 1 1</td>
</tr>
<tr>
<td>1 1 0 0 0 1 1</td>
<td>0 0 1 0 0 0 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1 1 0 0</td>
<td>1 1 1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 1 1 0 0</td>
<td>1 1 0 0 1 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1 2 3 3</td>
<td>1 0 1 1 0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>1 0 0 1 1 0 0 0</td>
</tr>
<tr>
<td></td>
<td>1 1 1 2 2 3 3 3 3</td>
</tr>
</tbody>
</table>

What to turn in:
1. A file including your program together with the results of your program for the three example circuits given in the writeup. Your program output should follow the format of the example output specified above.
2. A document detailing your evaluation of the efficiency of the data structures suggested in the writeup and providing a reasoning as to which one you have decided to use. The relevant parameters to consider in your selection should be primarily computational efficiency and secondarily space efficiency. This document needs also contain a detailed description as to how you implement the data structure in your code.
3. Your program source code files and a README file that explains how to compile and run your program.

**Turn-in method:**
Send all requested files to the TA via email. The email must be time-stamped by the deadline.

**Programming language:** C programs are preferred. If you are not familiar with C programming and want to use some other language, discuss with the TA about the possibility before you start the work.