CIS 350 Program 3 Fall 2005
Min Cut

Due: March 29nd
Purpose: Data Structure Design

You may work in teams of 1 – 2 members at no deduction.
Problem: Given a graph G , partition the vertices of the G in 2 equal sized subsets, so as to minimize the total number of edges cut.

Applications:
One important practical example of this problem is placing the components of an electronic circuit onto printed circuit boards so as to minimize the number of connection between boards. The components are the vertices of the graph, and the circuit connections are the edges. There is a maximum number of components which may be placed on any board. Since connections between boards have a high cost compared to connections within a board, the object is to minimize the number of interconnections between boards.
The partition problem also arises naturally in an attempt to improve the paging properties of programs for use in computers with paged memory organization. A program (at least statically) can be thought of as a set of connected entities. The vertices might be subroutines or procedure blocks or single instruction and data items depending on view point and the level of detail required. The connections between the vertices might represent possible flow or transfer of control or references from on entity to another. The problem is to assign the object to “pages” of a given size so as to minimize the number of references between objects which lie on different pages.
A final example might be the problem of a compiler of a multi-CPU computer, where the vertices are sections of code which are done sequentially and the edges represent order dependence between sections. The fewer edges cut, the fewer times the CPUs have to synchronize with each other.

Specifics:
Your graph will contain no loops but may have parallel edges. Graphs will be generated by random with the maximum degree of a vertex equal to the square root of the number of vertices. The number of vertices will always be even. The cost of your cut is the number of edges that go from one set to the other set.

Input

The program will input the name of a graph file. The graph file will have the number of vertices *n* on the first line. The rest of the file will consist of *n* lines. The *i*th line will begin with the degree of vertex *i* followed by the vertices that are adjacent to vertex *i*. Note that if *i* is adjacent to *j*, then *j* is adjacent to *i* . All numbers are separated by blanks.

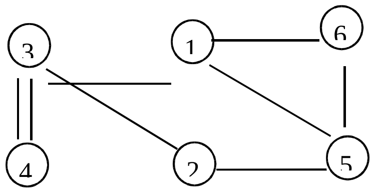
Output

The output will be the screen with the cost (number of edges cut the vertex numbers) and one set (separated by spaces) you put in 1 set into a file.

Input Graph file format:

#vertices
degreeOfVertex 1 vertices connected to vertex 1 (separated by a blank)
degree if vertex 2 vertices connected to vertex 2 (separated by a blank)
:

Example Input Graph file



6
3 2 6 5
4 4 4 1 3
2 2 5
2 2 2
3 1 3 6
2 1 5

Corresponding Screen Output

The min cost is 2 with the cut 1 5 6

Required Algorithm

Put the first *n*/2 vertices (1 .. *n/*2*)*in one set and the rest of the vertices (*n*/2+1 .. *n*) in the other set. For each vertex compute the inDegree (the number of edges that connect to vertices in the same set) and the outDegree (the number of edges that connect to vertices in the other set). Swap the vertices in each set with the largest (outDegree – inDegree), updating the vertices adjacent to the swapped vertices. In case of ties use numerical order. No longer consider those vertices.
Continue until at least one set of vertices does not have vertices positive outDegree-InDegree.
For example in the above
If we start with the sets {1, 2, 3} and {4, 5, 6}

vertex	inDegree	outDegree	outDegree - inDegree
1	1	2	1
2	1	1	0
3	2	2	0
4	0	2	2
5	1	2	1
6	1	1	0

So here we begin by swapping vertex 4 and vertex 1
So the sets become {2, 3, 4} and {1, 5, 6}

vertex	inDegree	outDegree	outDegree - inDegree
1	2	1	-1 LOCKED
2	1	1	0
3	3	1	-1
4	2	0	-2 LOCKED
5	2	1	-1
6	2	0	-2

Now there are no more productive vertices to swap so we are done.

Grading:
Grading (what to turn in)

What _ indicates program/other is memo	**Points**	**Due Tuesday March 29th**
External Documentation	**10**	
Your Name	1	
Description of the problem	2	
Input Specification	1	
Output Specification	1	
Algorithm Description UML	5	
Data Structure	**20**	
main data structure "structure"	1	
member functions / functions pre/post conditions for each	4	
Design of Data Structure Time and Space Efficiency and discussion and discussion of implementation	15	
Analysis	**15**	

What is the largest graph you can process in a reasonable amount of time. (analysis should be based on the # of vertices and the # of edges)	2	
Worst case time analysis for each function. (analysis should be based on the # of vertices and the # of edges)	4	
Worst case space analysis for each function(analysis should be based on the # of vertices and the # of edges)	4	
Test Plan	4	
Sample Runs	1	
Program Listing Style	**15**	
Your Name	1	
Description of the problem	2	
Variable Names	3	
Data Dictionary	2	
Pre/post conditions	3	
Length of functions	2	
Use of white space	2	
Functionality	**40**	
Main	10	
Inputs File Correctly	5	
Outputs Correctly	5	
Computes Correct Cut	20	