Report on the Implementation of the Kernighan-Lin Bi-Partition Algorithm

To compile: The command `make` in the presence of the implementation files will invoke the g++ compiler and linker as appropriate to generate the binary, `klpart`.

I. Data Structures

Three distinct data structure type were used in the implementation of the Kernighan-Lin bi-partition algorithm:

Arrays: I have used arrays in many places in my implementation of KL Bi-Partitioning (hereafter KLPart). Chiefly these include lists of D-values and G-values, and the representation of the partitions themselves.

Queues: In an attempt to mimic the pseudocode presented in the text, I opted to use a queue data structure to track partition swapping in the inner loops of KLPart.

Matrices: To store the cost matrix, I opted to use a 2n x 2n matrix (essentially a 2n x 2n array). Note that this matrix has size (#nodes)^2.

II. Pseudocode

The pseudocode of the algorithm is very similar to that of the text, and is briefly repeated here:

```
1. Partition vertices V into A, B st. A union B = V and A intersect B is empty.
2. Compute Dv for all vertices.
3. Empty swap queue and set iteration = 1, set A'=A and B'=B, empty array G
4. Find some node pair ai,bi st. gi = Dai+Dbi-2caibi is maximal
        Store this value of gi in array G
5. Add the pair ai,bi to the swap queue, and remove the pair from A' and B'
6. If A' and B' are not empty
        update Dv for remaining vertices
        increment iteration variable
        Goto 4
7. Find k st the sum of the first k values of array G is maximal, call this sum G_max
        If G_max > 0
                Make changes in swap queues permanent, and Goto 4
        Else STOP the algorithm
```

III. Structure and Organization

The call stack of the implementation follows graphically below. To narrate, the main function (in klmain.cpp) does some initial input parsing and checking, then begins keeping time. Main() then calls klpart() (in klpart.cpp), which performs the bipartitioning algorithm described above. Klpart() is responsible for reading its data from the disk; it does so by calling the readmatrix() function (in readmatrix.cpp). Throughout its execution, klpart() uses IntegerMatrix and IntegerQueue objects to organize its data (found in intmatrix.cpp and intqueue.cpp respectively).

```
int main()
     -Input parsing
     -Start time
     -Call klpart()
          -Call readmatrix()
          -Do algorithm as described above.
          -Return
     -Stop time
     -Print running time
     -Exit
```

IV. Running Time

The implementation's running time is summarized in the table below. The table is organized by OS, machine specifications, and running conditions. Note that for the runs on CAE Unix machines, the nice command was used (by request of CAE) to give remote and long-running jobs lower priority than real-time, and therefore likely negatively affected running time.

	MS-DOS, PIII 800MHz, 128MB main memory	UNIX (`niced`), sun-100.cae.wisc.edu
KL_2000_sparse.txt	138.279 seconds	318.354 seconds
KL_1000_dense.txt	747.214 seconds	1751.051 seconds

V. Improvements

It would be possible to improve (reduce) the memory requirements of the program by noting that the cost matrix is transitive, that is Cij = Cji for all i,j. Therefore, a complete 2n x 2n matrix is not needed to store the cost matrix, only n(n-1)/2 elements. However, for the simplicity of addressing a particular cost, the full 2n x 2n matrix was used. Note that the matrix memory requirement is still quadratic in the size of the input.
Similarly, it would be possible to improve the performance by eliminating certain data structures altogether: the G-array for instance. It would be more effective to simply track which combination has given a maximal gain, rather than record each permutation and then re-discover G_max.
Finally, the best way to speed performance of this implementation would be to eliminate the descriptive output. Most of the algorithm's running time is bound by I/O, and to eliminate intermediate (descriptive) I/O would be to cut running time dramatically.