Class	Description
class MatrixEntry	*This class represents a single cell in a matrix.*
uint32_t Geti()	*Returns a 'i' value.*
uint32_t Getj()	*Returns a 'j' value.*
uint32_t Getvalue()	*Returns a value field.*
uint32_t Setvalue(double newVal)	*Sets a value field with the new value.*
MatrixEntry(uint32_t iVal, uint32_t jVal, double val)	*Constructor.*
MatrixEntry(string valAsStr)	*Constructor that parses i,j,value fields from the string argument.*
string ToString()	*Return a string as comma separated values.*
void ParseValues(string valAsString)	*Parses a comma separated value string into MatrixEntry value i,j,value.*
class MatrixEntry::iComparer	*A sub-class that can be passes to std::algorithms to compare 'i' value.*
class MatrixEntry ::jComparer	*A sub-class that can be passes to std::algorithms to compare 'j' value.*

Class	Description
class IsingModel	*This class represents 'Ising Model' that can calculate energy.*
static double computeEnergy(*const SparseMatrix& m, const std::vector<int>& s*);	*As name suggests, it is used for computing the energy.* *Arguments:* • *m : matrix* • *s: s vector(either-1 or 1).* *Description:(based on Marshall's description)* */* computeEnergy: returns the energy of an Ising problem at a given state * * energy = (sum over i of h_i * s_i) + (sum over i, j of J_ij * s_i * s_j) * Treat diagonal entries of m (i.e. when m[a].i == m[a].j) as h values, * other entries as J. * * Do not assume anything about the contents of m. In particular, m is not * necessarily normalized. You may assume all entries of s are +1 or -1. */

Class	Description
class SMNormalizer	*This class represents a normalized sparse matrix.*
uint32_t GetMaxi()	*Returns the max i value in the matrix.*
uint32_t GetMaxj()	*Returns the max j value in the matrix.*
uint32_t GetMaxIndex()	*Returns the max index value (j>=i? return j:return i) in the matrix.*
SparseMatrix normalize(const SparseMatrix& m)	*Normalize the given matrix.* *Arguments:* • *SparseMatrix: A matrix.* *Description: (based on Marshall's description)* */* normalize: returns a "normalized" version of a SparseMatrix suitable for fast lookups.* * *Return value r must satisfy these conditions:* * *1. For any 0 <= a < r.size(): r[a].i <= r[a].j && r[a].value != 0* * *2. Unique, lexicographically-ordered indices. For any 0 <= a < b < r.size():* * *(r[a].i < r[b].i) || (r[a].i == r[b].i && r[a].j < r[b].j)* * *3. Same contents as input. For any pair i <= j, there is a single index a* * *such that r[a].i == i && r[a].j == j && r[a].value == v, where v is* * *the sum of all m[b].value where* * *(m[b].i == i && m[b].j == j) || (m[b].i == j && m[b].j == i), UNLESS* * *v == 0 (in accordance with condition 1).* */

Class	Description
Class DataHelper	*This class is useful for creating test data.*
static SparseMatrix CreateRandomMatrix(*uint32_t rows,* *uint32_t cols,* *uint32_t seedVal=rand(),* *bool bDumpToFile=false,* *string fileName="SparseMatrix",* *double fillDensityPercentage=30.0*)	*This method facilitates the creation of random matrix.* *Arguments:* • *rows : i Value that represents a row in matrix* • *cols: i Value that represents a col in matrix* • *seedVal: Seedvalue for the random generator.* • *bDumpToFile: flag that indicates whether to dump the randomly generated matrix to be stored on disk. The default file name is "SparseMatrix" and extension is always ".csv".* • *fileName : name of the csv file.* • *fillDensityPercentage: The % amount of non-zero elements.*
static SparseMatrix ReadMatrixFromFile()	*Reads matrix from file. To get the file name use GetRandomMatrixDiskFileName(). Following shows the sample of stored matrix contents in SparseMatrix.csv file.* *2,3,852.000000* *5,3,7747.000000* *6,6,3984.000000* *1,2,6965.000000*
static vector<int> CreateRandomS(uint32_t size = 10000);	*Creates a vector of given size and initializes it with random 1,-1 values.* *Arguments:* • *size : size of a vector.*
static string GetRandomMatrixDiskFileName()	*Reruns a file name where the last randomly generated matrix is stored or the default file name.*

Sample Code:

```
/**
 * Author: Vivek Trivedi
 * Date: July 29, 2016
 */
#include "SMNormalizer.h"
#include "IsingModel.h"
#include "DataHelper.h"
#include <iostream>
using namespace std;

//Sample code to compute the energy.
int main()
{
        //generate a random matrix.
      auto m = DataHelper::CreateRandomMatrix(1000, 1000);
      // instance of Sparse Matrix Normalizer.
      SMNormalizer sm;
        //feed the randomly generated matrix to SMNormalizer to normalize it.
      auto nsm = sm.normalize(m);
        //generate the 'S' vector to feed it to Ising energy calculation.
      auto vecS = DataHelper::CreateRandomS(sm.GetMaxIndex() + 1);
        //Compute the energy using Ising Model.
      cout << "\nEnergy : " << IsingModel::computeEnergy(nsm, vecS) << "\n";

      cout << "\nPress any key end to end...";
      getchar();
      return 0;
}
```

How to Compile:

First, Extract the **spWithTest.bz2** and follow the procedure depicted in the following screen-shot. Please ref. http://www.boost.org/doc/libs/1_61_0/more/getting_started/unix-variants.html for Boost installation steps.