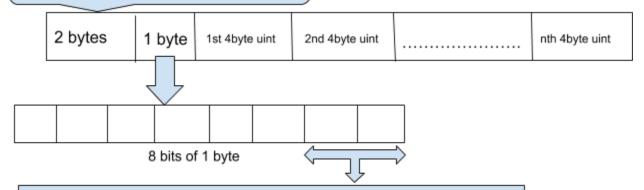
```
struct row {
    unsigned int rowid; //nodeid from the orig graph
    unsigned char *data; //pointer to adjacency list or adjacency
matrix row
    bool operator<(const struct row &a) const
    {
        return (this->rowid < a.rowid);
    }
};

/*
The *data pointer in row structure points to a char array that has following representation</pre>
```

First 2 bytes interpreted as "unsigned short int". Let us say value is v, then v = 1 + n



If second last bit is set to 1: every 4byte uint is a node-id, i.e., interpret it as adjacency list

If second last bit is set to 0: read the last bit.

If the last bit is set to 0, 1st 4byte uint represents num of consecutive 0s, 2nd 4byte uint as num of consecutive 1s, 3rd uint as consecutive 0s... and so on....

If the last bit is set to 1, 1st 4byte uint represents num of consecutive 1s, 2nd 4byte uint as num of consecutive 0s, 3rd uint as consecutive 1s... and so on...

This is called Run-Length-Encoding (RLE)

```
*/
typedef struct BitMat {
    list<struct row> bm;
    vector<struct row> vbm;
    unsigned int num_rows, num_totalBMs, num_columns, num_comm_so;
```

```
unsigned int row bytes, totalBMs bytes, column bytes,
common so bytes, dimension;
     unsigned long num triples;
     unsigned char *rowfold; //row_bytearr
     unsigned char *colfold; //column bytearr
     unsigned int last op;
     void freebm(bool list = true, bool vector = true)
           if (list) {
                for (std::list<struct row>::iterator it = bm.begin();
it != bm.end(); ) {
                     free((*it).data);
                     it = bm.erase(it);
                }
           }
           if (vector) {
                for (std::vector<struct row>::iterator it =
vbm.begin(); it != vbm.end(); ){
                     free((*it).data);
                     it = vbm.erase(it);
                }
           }
           if (rowfold != NULL) {
                free(rowfold);
                rowfold = NULL;
           }
           if (colfold != NULL) {
                free(colfold);
                colfold = NULL;
          num triples = 0;
     }
     void reset(void)
           for (std::list<struct row>::iterator it = bm.begin(); it
!= bm.end(); ){
                free((*it).data);
                it = bm.erase(it);
           }
```

```
if (rowfold != NULL) {
                memset(rowfold, 0, row_bytes);
          if (colfold != NULL) {
                memset(colfold, 0, column bytes);
          num_triples = 0;
     }
     ~BitMat()
          for (std::list<struct row>::iterator it = bm.begin(); it
!= bm.end(); ){
                free((*it).data);
                it = bm.erase(it);
          }
          if (rowfold != NULL)
                free(rowfold);
          if (colfold != NULL)
                free(colfold);
     }
} BitMat;
```