2020/02/19 - Bit Matrices (Part 1)

Monday, February 17, 2020 9:27 PM

Synopsis

- Back to using "~jplank" directories.
- This week is partial submission, due on WED, FEB 26.
- Test cases to hit: 1 - 28, 101 - 105, 107 - 120 (47 total)

Submission Command

- tar -cvf lab5_intermediate.tar src/bitmatrix.cpp

Suggested Order

> BitMatrix class
- Bitmatrix (int, int)
- Print
- Bitmatrix (const string &)
- Rows, Cols, Val (Access Trio)
- Write
- Set, Swap_Rows, R1_Plus_Equals_R2 (Modification Trio)
- PGM
- Copy

> BM_Hash class
- BM_Hash (int)
- Store, Recall
- All

Notes

- Note "M", a vector of C++ strings.
- "M" MUST only contain "0" or "1".
- When reading in, do it C++ style. Ask me why and I'll intimidate you with C-style getline.
- Most of these are self-explanatory functions. If in doubt, consult the comments in include/bitmatrix.hpp.
- For R1_Plus_Equals_R2,

```
'0' + '0' = '0'
'0' + '1' = '1'
'1' + '0' = '1'
'1' + '1' = '0'   // Overflow
```

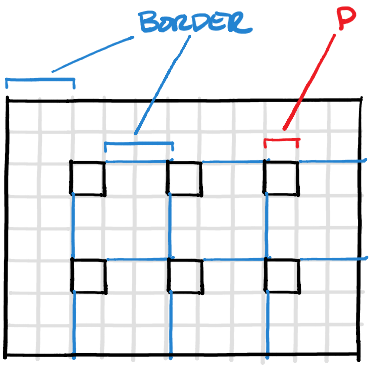
PGM

- Assume "border" is border width, "p" is width + height of each char in "M". Make 2D vector of unsigned char or int sized:
  - Width: border + ((p + border) x Cols)
  - Height: border + ((p + border) x Rows)

- Skip processing borders by just using vector.resize(size, default_value) like in previous labs.

(Ex.

border = 2
p = 1
Rows = 2
Cols = 3



11 x 8

w = 2 + ((1 + 2) x 3) = 11
h = 2 + ((1 + 2) x 2) = 8

- On M[i][j] = 1, pixel is 100
- On M[i][j] = 0, pixel is 255
- On border, pixel is 0.
- Format DOESN'T matter.

BM_HASH

- Create table of specific size.
- Use djb_hash from lecture notes.
- Separate Chaining is used, hence vector< vector< HTE > >.
  - STORE: Hash "key", then jump to spot in table (i = djb_hash(key) % Table.size()), and push_back if key doesn't already exist. If it does, obliterate its bm and replace.
  - RECALL: Figure it out. Should be obvious if you did "Store". Seriously.
- ALL: Go through all elements & push to new 1D vector. Then return it.

SEPARATE CHAINING DIAGRAM