

\*\*\*\*\* Cover Page \*\*\*\*\*

Class: CV  
Name: Frank Youmet  
Project: Project 5  
Project Name: Image Compression via Distance Transform  
Language: C++  
Due Date: 10/20/2024 before 12:00AM  
Submit Date: 10/20/2024 before 9:00PM

---

## Top Level algorithm steps

---

\*\*\*\*\*

VI. main (...)

\*\*\*\*\*

step 0:        inFile, prettyPrintFile, skeletonFile, deCompressedFile, logFile ← open via argv []  
               numRows, numCols, minVal, maxVal ← read from inFile  
               dynamically allocate ZFAry and skeletonAry with extra 2 rows and 2 cols  
               distanceChoice ← get from argv [2]

Step 1:        setZero (ZFAry)  
               setZero (skeletonAry)

Step 2:        loadImage (inFile, ZFAry)  
               prettyPrint (ZFAry, prettyPrintFile) // with caption "\*\*\* Below is input image\*\*\*"

Step 3:        distanceTransform (ZFAry, distanceChoice, prettyPrintFile, logFile)

Step 4:        compression (ZFAry, distanceChoice, skeletonAry, skeletonFile, prettyPrintFile, logFile)

Step 5:        close skeletonFile

Step 6:        reopen skeletonFile

Step 7:        setZero (ZFAry)

Step 8:        loadSkeleton (skeletonFile, ZFAry, logFile)  
               prettyPrint (ZFAry, prettyPrintFile) // with caption "\*\*\* Below is the loaded skeletonwith choice =\*\*\*"

Step 9:        deCompression (ZFAry, distanceChoice, prettyPrintFile, logFile) // Perform decompression

Step 10:       deCompressedFile ← output numRows, numCols, minVal, maxVal

Step 11:       binThreshold (ZFAry, deCompressedFile)

Step 12: close all files

\*\*\*\*\*

### Source Code

\*\*\*\*\*

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
using namespace std;
```

```

class DistanceSkeleton{
public:
    int numRows;
    int numCols;
    int minVal;
    int maxVal;
    int newMinVal;
    int newMaxVal;
    int** ZFAry;
    int** skeletonAry;
    int distanceChoice;

    void setZero(int** Ary){
        for (int i = 0; i < (numRows + 2); i++){
            for (int j = 0; j < (numCols + 2); j++){
                Ary[i][j] = 0;
            }
        }
    }

    void loadImage(ifstream& inFile, int** ZFAry){
        for(int i = 1; i < (numRows + 1); i++){
            for (int j = 1; j < (numCols + 1); j++){
                inFile >> ZFAry[i][j];
            }
        }
    }

    void prettyPrint(int** ZFAry, ofstream& prettyPrintFile){
        for (int i = 0; i < (numRows + 2); i++){
            for (int j = 0; j < (numCols + 2); j++){
                prettyPrintFile << ZFAry[i][j] << " ";
            }
            prettyPrintFile << "\n";
        }
        prettyPrintFile << "\n";
    }

    void distanceTransform(int** ZFAry, int distanceChoice, ofstream& prettyPrintFile, ofstream& logFile){
        logFile << "Entering DistanceTransform() method";
        logFile << "\n";

        distancePass1(ZFAry, distanceChoice, logFile);
        prettyPrintFile << "1st pass distance transform with choice = " << distanceChoice << "\n";
    }
}

```



```

        ZFAry[i][j] = min(min(min(ZFAry[i][j], ZFAry[i][j] + 1) + 1), ZFAry[i + 1][j - 1] + 2), min(ZFAry[i
+ 1][j] + 1, ZFAry[i + 1][j + 1] + 2));
        break;

        default:
            break;
    }
}
}
}
}
}

```

```

void compression(int** ZFAry, int distanceChoice, ofstream& skeletonFile, ofstream& prettyPrintFile,
ofstream& logFile){
    logFile << "Entering compression() method.";
    logFile << "\n";

    computeLocalMaxima(ZFAry, skeletonAry, distanceChoice, logFile);
    prettyPrintFile << "Local maxima, skeletonAry with choice = " << distanceChoice;
    prettyPrintFile << "\n";
    prettyPrint(skeletonAry, prettyPrintFile);

    extractSkeleton(skeletonAry, skeletonFile, logFile);
    logFile << "In compression() Below is skeleton Array with choice = " << distanceChoice;
    logFile << "\n";
    prettyPrint(skeletonAry, logFile);

    logFile << "Leaving compression() method";
    logFile << "\n";
}

```

```

void computeLocalMaxima(int** ZFAry, int** skeletonAry, int distanceChoice, ofstream& logFile){
    for(int i = 1; i < (numRows + 1); i++){
        for (int j = 1; j < (numCols + 1); j++){
            if(isLocalMaxima(ZFAry, i, j)){
                skeletonAry[i][j] = ZFAry[i][j];
            }
            else{
                skeletonAry[i][j] = 0;
            }
        }
    }
}
}
}

```

```

bool isLocalMaxima(int** ZFAry, int i, int j) {
    int currentVal = ZFAry[i][j];

    return (currentVal >= ZFAry[i-1][j-1] &&
            currentVal >= ZFAry[i-1][j] &&
            currentVal >= ZFAry[i-1][j+1] &&
            currentVal >= ZFAry[i][j-1] &&
            currentVal >= ZFAry[i][j+1] &&
            currentVal >= ZFAry[i+1][j-1] &&
            currentVal >= ZFAry[i+1][j] &&
            currentVal >= ZFAry[i+1][j+1]);
}

```

```

void extractSkeleton(int** skeletonAry, ofstream& skeletonFile, ofstream& logFile){
    skeletonFile << numRows << " " << numCols << " " << minVal << " " << maxVal;
    skeletonFile << "\n";
    for(int i = 0; i < (numRows + 2); i++){
        for(int j = 0; j < (numCols + 2); j++){
            if(skeletonAry[i][j] > 0) {
                skeletonFile << i << " ";
                skeletonFile << j << " ";
                skeletonFile << skeletonAry[i][j];
                skeletonFile << "\n";
            }
        }
    }
}

```

```

void loadSkeleton(ifstream& skeletonFile, int** ZFAry, ofstream& logFile){
    int i;
    int j;
    int val;

    string line;
    getline(skeletonFile, line);
    while(getline(skeletonFile, line)){
        istringstream iss(line);
        iss >> i;
        iss >> j;
        iss >> val;

        ZFAry[i][j] = val;
    }
}

```

```

void deCompression(int** ZFAry, int distanceChoice, ofstream& prettyPrintFile, ofstream& logFile){
    logFile << "Entering deCompression() method";
    logFile << "\n";

    expansionPass1(ZFAry, distanceChoice, logFile);
    prettyPrintFile << "1st pass expansion with choice = " << distanceChoice;
    prettyPrintFile << "\n";
    prettyPrint(ZFAry, prettyPrintFile);

    expansionPass2(ZFAry, distanceChoice, logFile);
    prettyPrintFile << "2nd pass expansion with choice = " << distanceChoice;
    prettyPrintFile << "\n";
    prettyPrint(ZFAry, prettyPrintFile);
}

```

```

void expansionPass1(int** ZFAry, int distanceChoice, ofstream& logFile){
    for(int i = 1; i < (numRows + 1); i++){
        for(int j = 1; j < (numCols + 1); j++){
            if(ZFAry[i][j] == 0){
                int maxVal = ZFAry[i][j];
                switch (distanceChoice)
                {
                    case 8:
                        for (int row = i - 1; row <= i + 1; row++) {
                            for (int col = j - 1; col <= j + 1; col++) {
                                maxVal = max(maxVal, ZFAry[row][col]-1);
                            }
                        }
                        ZFAry[i][j] = maxVal;
                        break;
                    case 4:
                        ZFAry[i][j] = max(max(max(max(max(ZFAry[i][j], ZFAry[i][j] - 1)-1),
                        ZFAry[i][j] + 1]-1),
                        ZFAry[i - 1][j] - 1]-2),
                        ZFAry[i - 1][j]-1),
                        max(ZFAry[i - 1][j] + 1]-2,
                        max(ZFAry[i + 1][j] - 1]-2,
                        max(ZFAry[i + 1][j]-1,
                        ZFAry[i + 1][j] + 1]-2)))));
                        break;
                    default:
                        break;
                }
            }
        }
    }
}

```

```

    }
  }
}
}

```

```

void expansionPass2(int** ZFAry, int distanceChoice, ofstream& logFile){
  for(int i = (numRows); i > 0 ; i--){
    for(int j = (numCols); j > 0 ; j--){
      int maxVal = ZFAry[i][j];
      switch (distanceChoice)
      {
        case 8:
          for (int row = i - 1; row <= i + 1; row++) {
            for (int col = j - 1; col <= j + 1; col++) {
              maxVal = max(maxVal, ZFAry[row][col]-1);
            }
          }
          ZFAry[i][j] = maxVal;
          break;
        case 4:
          ZFAry[i][j] = max(max(max(max(max(ZFAry[i][j], ZFAry[i][j] - 1)-1),
            ZFAry[i][j] + 1]-1),
            ZFAry[i - 1][j] - 1]-2),
            ZFAry[i - 1][j]-1),
            max(ZFAry[i - 1][j] + 1]-2,
            max(ZFAry[i + 1][j] - 1]-2,
            max(ZFAry[i + 1][j]-1,
            ZFAry[i + 1][j] + 1]-2))));
          break;
        default:
          break;
      }
    }
  }
}

```

```

void binThreshold(int** ZFAry, ofstream& deCompressedFile){
  for(int i = 0; i < (numRows + 2); i++){
    for(int j = 0; j < (numCols + 2); j++){
      if(ZFAry[i][j] >= 1){
        deCompressedFile << "1 ";
      }
      else{

```

```

        deCompressedFile << "0 ";
    }
}
deCompressedFile << "\n";
}
}

```

```
};
```

```
int main(int argc, char** argv){
```

```
    //Creates an instance of each file passed as an argument and makes sure they can be opened.
```

```
    ifstream inFile(argv[1]);
```

```
    if(!inFile.is_open()){
```

```
        cout << "Unable to open: " << argv[1];
```

```
        exit(1);
```

```
    }
```

```
    ofstream prettyPrintFile(argv[3]);
```

```
    if(!prettyPrintFile.is_open()){
```

```
        cout << "Unable to open: " << argv[3];
```

```
        exit(1);
```

```
    }
```

```
    ofstream skeletonFile(argv[4]);
```

```
    if(!skeletonFile.is_open()){
```

```
        cout << "Unable to open: " << argv[4];
```

```
        exit(1);
```

```
    }
```

```
    ofstream deCompressedFile(argv[5]);
```

```
    if(!deCompressedFile.is_open()){
```

```
        cout << "Unable to open: " << argv[5];
```

```
        exit(1);
```

```
    }
```

```
    ofstream logFile(argv[6]);
```

```
    if(!logFile.is_open()){
```

```
        cout << "Unable to open: " << argv[6];
```

```
        exit(1);
```

```
    }
```



```

//Creates an instance of the distance skeleton class.
DistanceSkeleton* distanceSkeleton = new DistanceSkeleton;

//Extracts the numRows, numCols, minVal, and maxVal values from the input file.
inFile >> distanceSkeleton->numRows;
inFile >> distanceSkeleton->numCols;
inFile >> distanceSkeleton->minVal;
inFile >> distanceSkeleton->maxVal;

//Dynamically allocates ZFAry and skeletonAry to have 2 extra rows and columns.
distanceSkeleton->ZFAry = new int*[distanceSkeleton->numRows + 2];
distanceSkeleton->skeletonAry = new int*[distanceSkeleton->numRows + 2];
for(int i = 0; i < distanceSkeleton->numRows + 2; i++){
    distanceSkeleton->ZFAry[i] = new int[distanceSkeleton->numCols + 2];
    distanceSkeleton->skeletonAry[i] = new int[distanceSkeleton->numCols + 2];
}

//Sets distanceChoice to the second argument passed.
distanceSkeleton->distanceChoice = atoi(argv[2]);

//Sets all values of ZFAry and skeletonAry to zero.
distanceSkeleton->setZero(distanceSkeleton->ZFAry);
distanceSkeleton->setZero(distanceSkeleton->skeletonAry);

//Loads the image from the inFile into the ZFAry
distanceSkeleton->loadImage(inFile, distanceSkeleton->ZFAry);

distanceSkeleton->prettyPrint(distanceSkeleton->ZFAry, prettyPrintFile);

distanceSkeleton->distanceTransform(distanceSkeleton->ZFAry, distanceSkeleton->distanceChoice,
prettyPrintFile, logFile);

distanceSkeleton->compression(distanceSkeleton->ZFAry, distanceSkeleton->distanceChoice,
skeletonFile, prettyPrintFile, logFile);

skeletonFile.close();

ifstream skeletonInputFile(argv[4]);
if(!skeletonInputFile.is_open()){
    cout << "Unable to open: " << argv[4] << " for reading";
    exit(1);
}

```







[illegible]

```
40 22 0 1
11 12 10
30 12 10
```

### deCompressedFile for img1 using 4-distance

[illegible]

### logFile for img1 using 4-distance

[illegible]

### PrettyPrint for img1 using 8-distance

[illegible][illegible]

[illegible]



Below is the loaded skeleton with choice = 8

[illegible]

```
1st pass expansion with choice = 8
```

[illegible]

### SkeletonFile for img1 using 8-distance

### deCompressedFile for img1 using 8-distance

[illegible]

### logFile for img1 using 8-distance

[illegible]

### PrettyPrint for img2 using 8-distance

[illegible][illegible]

[illegible]

[illegible][illegible]

[illegible]

SkeletonFile for img2 using 8-distance

49 64 0 1  
4 31 1  
6 31 2  
8 31 3  
10 31 4  
12 31 5  
13 22 1  
13 24 2  
13 26 3  
13 28 4  
13 30 5  
13 31 5  
13 32 5  
13 34 4  
13 36 3  
13 38 2  
13 40 1  
14 31 5  
14 46 1  
14 62 1  
15 46 1  
15 47 1  
15 61 1  
15 62 1  
16 31 4  
17 47 2  
17 61 2  
18 31 3  
18 47 2  
18 48 2  
18 60 2  
18 61 2  
20 31 2  
20 48 3  
20 60 3  
21 9 6  
21 10 6  
21 11 6  
21 12 6  
21 13 6  
21 14 6  
21 15 6  
21 48 3  
21 49 3  
21 59 3  
21 60 3  
22 31 1  
23 31 1  
23 49 4  
23 51 3  
23 53 2  
23 54 2  
23 55 2  
23 67 3  
23 69 4  
25 31 2  
25 48 3  
25 49 3  
25 59 3  
25 60 3  
26 48 3  
26 60 3  
27 31 3  
28 47 2  
28 48 2  
28 60 2  
28 61 2  
29 31 4  
29 47 2  
29 61 2  
31 31 5  
31 46 1  
31 47 1  
31 61 1  
31 62 1  
32 22 1  
32 24 2  
32 26 3  
32 28 4  
32 30 5  
32 31 5  
32 32 5  
32 34 4  
32 36 3  
32 38 2  
32 40 1  
32 46 1  
32 62 1  
33 31 5  
35 31 4  
37 31 3  
39 31 2  
41 31 1

---



[illegible][illegible]