Darrell Percey

Data Structure II

Project 4

Functional Decomposition

**Data Structures**

//Holds link list of all packaging

struct information{

binP OnFF;

binP OnNF;

binP OnBF;

binP OffFF;

binP OffBF;

int sizeList;

int maxSize;

float totalList[LISTSIZE];

} information;

//Bins for each set

struct bin{

int totalItems;

float currentSize;

float itemList[LISTSIZE];

binP next;

} bin;

The first struct information holds all the bin linked list for each of the packing types. Then the bins hold the items and current bin sizes.

**Files and Functions**

**Test.c**

Runs the program from the OfflineSet and OnlineSet then uses the Builds file to print the information and free the memory of all the list.

**OfflineSet.c / OfflineSet.h**

/\*

\* Function: offlineFirstFit()

\* Return: binP

\*

\* Description:

\* Does the first fit bin packing. Similar to

\* online but this one sorts the array from small

\* to large then places in the bins

\*

\*/

Does the bin packing for a first fit algorithm in the offline style.

/\*

\* Function: offlineBestFit()

\* Return: binP

\*

\* Description:

\* Similar to the online best fit but this one.

\* first sorts the array from largest to smallest.

\* Once sorted then it tries to calculate the best

\* bins that each item will fit in.

\*

\*/

Does the bin packing for a best fit algorithm in the offline style.

/\*

\* Function: sortSetFF()

\* Return: float\*

\*

\* Description:

\* Sorts the array from smallest to largest for the

\* first fit offline bin packing. This allows it to

\* place in the bins more efficiently.

\*

\*/

Sorts the items in the array for the first fit bin packing algorithm

/\*

\* Function: sortSetBF()

\* Return: float\*

\*

\* Description:

\* Sorts for the offline best fit bin packing

\* It sorts the items in the bin from largest

\* to the smallest for processing.

\*

\*/

Sorts the items for the best fit bin packing algorithm with for offline.

**OnlineSet.c / OnlineSet.h**

/\*

\* Function: onlineFirstFit()

\* Return: binP

\*

\* Description:

\* Does the first fit bin packing. Reads from

\* the array one at a time finding the first

\* avaliable bin to place

\*

\*/

Does the algorithm for the first fit online bin packing.

/\*

\* Function: onlineNextFit()

\* Return: binP

\*

\* Description:

\* Does the next fit bin packing. Reads from

\* array and tries to place in current bin.

\* If the current bin is full it makes a new bin.

\*

\*

\*/

Does the algorithm for the next fit online bin packing.

/\*

\* Function: onlineBestFit()

\* Return: binP

\*

\* Description:

\* Does the best fit packing. Reads from the

\* array and tries to calcualte which bin is best.

\* It does so by comparing if it can fit and if the

\* current size would pack it more.

\*

\*/

Does the best fit online bin packing algorithm.

**Builds.c / Builds.h**

/\*

\* Function: createNewInfoNode()

\* Return: infoP

\*

\* Description:

\* Creates a new node for the information structure

\* which holds all the bins for each of the packing

\* algorithms.

\*

\*/

Creates the information structure to hold all the linked list, list of items, and max bin size.

/\*

\* Function: readFileInfo()

\* Return: infoP

\*

\* Description:

\* Reads in the bin sizes from the top of the file

\* then all the following will be floats for items.

\* These are read into an array for easier handling.

\*

\*/

Reads the information from the file “theItems.txt” and places it in the information structure.

/\*

\* Function: createNewBinNode()

\* Return: binP

\*

\* Description:

\* Creates a new bin and works will all the algorithms

\* FF, BF, NF both online and offline.

\*

\*/

Makes bins for the link list of each algorithm.

/\*

\* Function: addToBin()

\* Return:

\*

\* Description:

\* Adds an item to the bin, works will all bins.

\* Increases bin size, the total items in the bin.

\* and places the item into the array for look-up.

\*

\*/

Adds the a new item to the bin.

/\*

\* Function: printAllInfo()

\* Return:

\*

\* Description:

\* Prints the table with the amount of bins used by

\* each of the bin packings and then displays the

\* size of each bin and the items in them.

\*

\*/

Prints all the information to the screen. Makes a table display and then prints the bins information.

/\*

\* Function: displayInformation()

\* Return:

\*

\* Description:

\* Prints the bins size and items in each bin for the

\* passed in bin packing head structure.

\*

\*/

/\*

\* Function: calculateBinsUsed()

\* Return: int

\*

\* Description:

\* Finds the amount of bins used for the packing then

\* returns the valuve to be displayed.

\*

\*/

Calculate the amount of bins used for each algorithm.

**Conclusion**

Depending on the numbers you can see the different in the packing algorithm optimal finding. However, you can see for large amounts of items then the complexity could go up with timing. I feel like next fit would be faster than the other two due to it not having to search other bins.