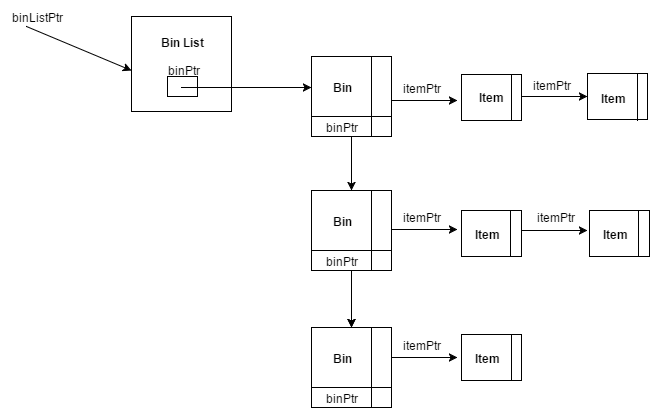
Gedd Johnson

Project 4 Bin Packing

Data Structures and Algorithms II

**Data Structures**



**Figure 1.** Block diagram of the adjacency list data structure used for this project.

struct BinList{

int numBins;

BinP head;

};

struct Bin{

int capacity;

int currentSize;

BinP nextBin;

ItemP firstItem;

};

struct Item{

int size;

ItemP nextItem;

};

**Figure 2.** C code showing the struct for each data structure within the adjacency list.

For this project I elected to use an adjacency list data structure to represent the bins and the items within them. My implementation, shown in Figure 1, contains a Bin List ADT that is accessed by main(), which performs all algorithms on the Bin List. Inside the Bin List is a Bin ADT that points to other Bins via a binPtr and also to Item ADTs via an itemPtr. The Items then form a linked list outside the Bin such that the Bin points to the first item in the list, which is also the first Item placed into the Bin. Each Bin has a capacity and a currentSize. As Items are added, their size is added to the currentSize of the Bin and they added to the end linked list of Items pointed to by its respective Bin. In this way, I am able to keep track of exactly how much space each item is using and I can traverse the Bins to see their contents.

I chose this data structure because I believe it is a very intuitive way of representing the Bins and Items. Moreover, this data structure has a simple implementation as it is basically a linked list. I considered other ways of representing the data such as a 2-dimensional array; however, I believe that the adjacency list is a more portable and maintainable design. For instance, the adjacency list can keep growing as large as it needs because new nodes, whether Bins or Items, are simply added onto the ends of the list. Furthermore, by abstracting Bins and Items I have made those ADTs portable to other programs.