CS575 Project 3

Due at 11:59pm on April 26
(Submit through blackboard)

1. [60%] Implement the following 0/1 knapsack algorithms for maximum 10 items.

    1) [5%] Implement the brute force method to solve the 0/1 knapsack problem. Show the final solution. Specifically, (a) print the total profit and weight; and (b) print the selected items.
    Implement the code in **bruteForce.c (or bruteForce.cpp)** file, make command should be able to compile the code and generate **bruteForce.out** executable file. Input file will be passed as command line argument like: **./bruteForce.out input.csv**

    2) [15%] Implement the dynamic programming algorithm for 0/1 knapsack: (a) print the total profit and weight; and (b) print the selected items. Compare the result to the result you have got in 1). If you have implemented the dynamic programming algorithm correctly, the total profit achieved by this backtracking method must be equal to that achieved by the brute force method implemented in 1).

    Implement the code in **dynamic.c** file, make command should be able to compile the code and generate **dynamic.out** executable file. Input file will be passed as command line argument like: **./dynamic.out input.csv**

    3) [40%] Implement the backtracking algorithm for 0/1 knapsack: (a) print the total profit and weight; and (b) print the selected items. Compare the result to the result you have got in 1). If you have implemented the backtracking algorithm correctly, the total profit achieved by this backtracking method must be equal to that achieved by the brute force method implemented in 1).

    Implement the code in **backtrack.c** file, make command should be able to compile the code and generate **backtrack.out** executable file. Input file will be passed as command line argument like: **./backtrack.out input.csv**

    CSV input file will contain the number of elements in the first line, knapsack capacity in the second line, profits in the third line, and weights in the fourth line.

(The maximum number of elements is 10 as specified above.) Note that this just an example data. Your programs should work correctly for any input:

```
3
35
50, 140, 60
5, 20, 15
```

2. [30%] Implement the Huffman code algorithm discussed in class. Use the following table as a sample. In the table below, the first two rows are the input, and the last row is the output of the Huffman code algorithm. The third row is just an example of fixed-length code: It is neither input to nor output from Huffman. (Your program should work correctly for any input; however, assume that you have no more than 20 characters.)

char	a	b	c	d	e	f	total bits
#	45	13	12	16	9	5	
fixed	000	001	010	011	100	101	300
variable	0	101	100	111	1101	1100	224

Implement the code in **huffman.c** file, make command should be able to compile the code and generate huffman.out executable file. Input file will be passed as command line argument like: **./huffman.out huffman.csv**

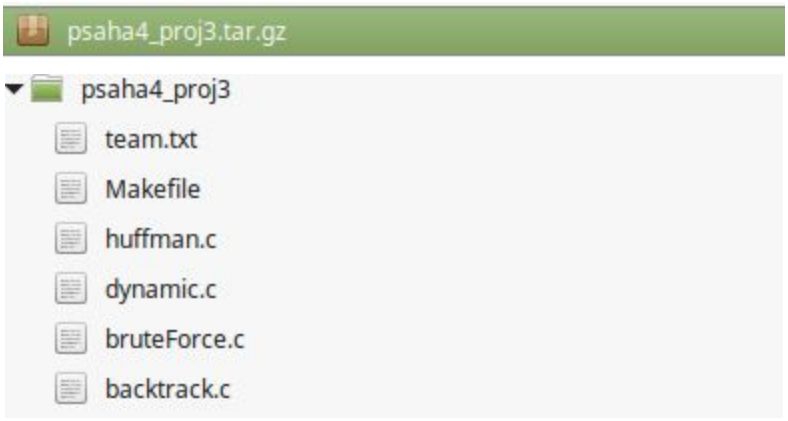
Input file will contain data in csv format. The input file for the example above is:
a, b, c, d, e, f
45, 13, 12, 16, 9, 5

3. [10%] 10% is reserved for good coding style, meaningful comments, and correctly following the submission guidelines below.

Submission Guidelines:

- Please submit a tar.gz file with the below naming convention. <user name>_project3.tar.gz. If you are working as group, then only mention the user's name who is going to submit the project in blackboard. "team.txt" should contain only the team members full name.

- Make sure you include all the source code files as mentioned below:
  Put all your source code files inside a directory <user name>_project3 and then create a tarball with name <user name>_project3.tar.gz
  Do not add any extra directory levels.



- Your makefile should be able to compile all the source code and generate .out files. make sure you makefile will not run the executable files.

- You should run and test your code in **remote.cs.binghamton.edu** and make sure it works there without any issues. You code will be evaluated in the same environment. If code does not run on remote server the TA will not run it in any other environments.

- Wrong directory structure and presence of unnecessary files will cost you points.

- Do not create unnecessary output files, header files, printf statements that are not asked to create.

- Please do not assume anything, email questions to the TA (psaha4@binghamton.edu), and see him during his office hours for any required clarifications.

Important Policies:

- **Project Group:** Up to two students can work as a team; however, each team should work independently of the other teams. Make sure you don't look into others' code -- each team should write its code on its own. **Copying from the Internet or any other online/offline sources is cheating and, therefore, it is strictly prohibited.** Note that **plagiarism** will be thoroughly checked using automated tools. If automated tools find any cheating/plagiarism, **you will get zero point for this project, your final letter grade will be decreased by one level, and you have to fill in and sign the official form that will be kept by the Binghamton University.**
- Your code will be compiled and executed. **If your code does not compile or produce any runtime errors such as segmentation faults or bus errors, you will get zero.** Before submitting your code, run it in a Linux machine (e.g., a remote machine) to remove any such errors. You can use "**valgrind**" (http://valgrind.org/) to debug memory errors, such as segmentation faults or bus errors. Note that **robust programming** is essential for developing high quality software. If you are curious about robust programming, check this out: http://nob.cs.ucdavis.edu/bishop/secprog/robust.html
- **Assistance: You are supposed to figure out how to implement this project. The instructor and TA will be more than happy to explain fundamental algorithms covered in class; however, they will not tell you how to implement them or help you to make any kind of decision. Neither will they read or debug your code.**
- By the same token, **the instructor and TA will not take a look at an emailed code. Neither will they answer any emailed questions regarding implementation**. If you have questions about general C/C++ programming or Linux, use Google search, which will be much faster than asking the TA. (Don't copy from the Internet though.) Contact the **TA Pankaj Saha** (psaha4@binghamton.edu) and meet during the office hours for any other required clarification.