**CPSC 335**
**Fall 2014**
**Project #2 — exhaustive search**

## Introduction

In this project you will reduce the problem of selecting which Debian Linux packages to include on installation media, to the classical *knapsack problem*. You will implement an exhaustive search algorithm for this problem, and measure its performance.

## The hypotheses

This experiment will test two hypotheses:

1. Exhaustive search algorithms are feasible to implement and produce correct outputs.
2. Algorithms with exponential (e.g. $O(2^n)$) running times are extremely slow, probably too slow to be of practical use.

## A practical knapsack problem

Debian (`http://debian.org`) is a well-established GNU/Linux distribution. Ubuntu Linux, arguably the most popular Linux distribution, is based on Debian. Debian software is organized into *packages*. Each package corresponds to a software component, such as the GCC compiler or Firefox web browser. There are packages not only for the software that comprises the operating system, but also end-user applications. Nearly every piece of noteworthy open source software has a corresponding package. There are over 36,000 Debian source code packages available.

This wide selection of packages is mostly a good thing. However it poses a problem for creating *installation media* – the floppy disks, CDs, DVDs, or flash drive images that administrators use to install the operating system. If a user wants to install a package that is missing from their installation media they need to download it over the Internet, so it is beneficial to include as many packages as possible. It is impractical to include every package on those media, so the Debian project needs to select a small subset of important packages to include.

Another part of Debian is the Popularity Contest, or *popcon:*

`http://popcon.debian.org/`

The Popularity Contest tries to measure the popularity of each Debian package. Users may elect to participate, in which case the list of their installed packages is transmitted back to Debian. These lists are tallied as *votes*. So, thanks to popcon, we can get a vote tally for each package.

In this project, you will write code that solves the following problem:

The *optimal package set* problem is:

**input:** a list of $n$ Debian packages, each with a size (in kilobytes) and number of votes, both of which are integers; and an integer $W$ representing the capacity of the install media (in kilobytes)
**output:** a list of packages with total size $\leq W$, such that the total number of package votes is maximized
**size:** $n, W$

You should recognize this as a thinly-veiled instance of the *knapsack problem.* Our package votes correspond to knapsack values, and our package sizes correspond to knapsack weights.

#### The input

I have created a text file called `packages.txt`. This file contains the name, size, and votes for approximately 36,000 binary packages that are available on the amd64 platform and have popcon information. The file obeys the following

format:

```
[number of packages]
[name 0][space][votes 0][space][size 0][newline]
[name 1][space][votes 1][space][size 1][newline]
[name 2][space][votes 2][space][size 2][newline]
...
```

The packages are sorted from the largest number of votes to the smallest. I used this format because I expect it will be easy to parse.

Here are the first five lines of `packages.txt`:

```
36149
debianutils 128329 90
libgcc1 128327 46
dpkg 128294 2672
perl-base 127369 1969
```

This file is posted in TITANium along with this requirements document.

#### Creating problem instances

We will create knapsack problem instances of varying input sizes $n$ by using the first $n$ entries in `packages.txt`. In other words, to create a problem instance with $n = 100$, only use the first $100$ packages listed in the file as input to your algorithm.

#### The algorithm

Implement an exhaustive search solution to the problem. We developed an exhaustive search algorithm for knapsack in class; our pseudocode had the following structure.

```
def ExhaustiveKnapsack(package_list, W):
    best = None
    for candidate in subsets(package_list):
        if verify_knapsack(candidate, W):
            if compare_knapsack(candidate, best):
                best = candidate
    return best
```

The definitions of the verify_knapsack and compare_knapsack parts followed directly from the knapsack problem definition, and are straightforward.

You can use either of the two algorithms described in the lecture notes for enumerating power sets. Alternatively, you may use the powerset function implemented in the Python itertools documentation.

**Implementation and empirical analysis**

Implement your algorithm in Python 3. As with Project 1, you must write your own code. The knapsack algorithm should be encapsulated in its own function. You should have a separate main type function that runs the knapsack code for various values of $n$ and collects accurate timing data.

You will need to write the following code:

1. Code to load the packages file into memory. I recommend defining a class called DebianPackage (or similar) that represents one package with fields for a name, size, and vote count.
2. A test harness program similar to the one in the previous project. The harness must accommodate variable values of $n$ and $W$.
3. The exhaustive search knapsack algorithm. The algorithm should take a list of package objects, and values $n$ and $W$ as arguments, and return a list of package objects.
4. Code to print out solutions. A solution is a list of packages, so your code

should print out the name, votes, and size of each package, as well as the total votes and total size of the solution.

## Sample output

```
--- n=20 W=1000
  -- Exhaustive search solution --
     debianutils 90 128329
     libgcc1 46 128327
     debconf 168 121503
     gzip 142 121346
     lsb-base 26 121121
     sysv-rc 80 121092
     sysvinit-utils 116 121078
     base-files 65 121072
     initscripts 93 121037
     libselinux1 109 120918
     total size=935 total votes=1225823
     Elapsed time = 1.82 seconds
--- n=24 W=720
  -- Exhaustive search solution --
     debianutils 90 128329
     libgcc1 46 128327
     gzip 142 121346
     lsb-base 26 121121
     sysv-rc 80 121092
     sysvinit-utils 116 121078
     base-files 65 121072
     initscripts 93 121037
     base-passwd 49 120856
     total size=707 total votes=1104258
     Elapsed time = 31.52 seconds
--- n=24 W=1440
  -- Exhaustive search solution --
     debianutils 90 128329
     libgcc1 46 128327
     debconf 168 121503
     gzip 142 121346
     sed 261 121220
     lsb-base 26 121121
     sysv-rc 80 121092
     sysvinit-utils 116 121078
     base-files 65 121072
     initscripts 93 121037
     libselinux1 109 120918
     cron 120 120872
     base-passwd 49 120856
     total size=1365 total votes=1588771
     Elapsed time = 31.37 seconds
--- n=24 W=25600
  -- Exhaustive search solution --
     debianutils 90 128329
```

```
    libgcc1 46 128327
    dpkg 2672 128294
    perl-base 1969 127369
    debconf 168 121503
    grep 595 121426
    gzip 142 121346
    login 980 121332
    coreutils 6505 121240
    bash 1673 121229
    sed 261 121220
    findutils 805 121173
    lsb-base 26 121121
    sysv-rc 80 121092
    sysvinit-utils 116 121078
    base-files 65 121072
    initscripts 93 121037
    util-linux 846 121022
    mount 272 120961
    libselinux1 109 120918
    cron 120 120872
    base-passwd 49 120856
    apt 1356 120826
    tzdata 488 120813
    total size=19526 total votes=2934456
    Elapsed time = 41.78 seconds
--- n=24 W=65536
 -- Exhaustive search solution --
    debianutils 90 128329
    libgcc1 46 128327
    dpkg 2672 128294
    perl-base 1969 127369
    debconf 168 121503
    grep 595 121426
    gzip 142 121346
    login 980 121332
    coreutils 6505 121240
    bash 1673 121229
    sed 261 121220
    findutils 805 121173
    lsb-base 26 121121
    sysv-rc 80 121092
    sysvinit-utils 116 121078
    base-files 65 121072
    initscripts 93 121037
    util-linux 846 121022
    mount 272 120961
    libselinux1 109 120918
    cron 120 120872
    base-passwd 49 120856
    apt 1356 120826
    tzdata 488 120813
    total size=19526 total votes=2934456
    Elapsed time = 41.89 seconds
```

## Deliverables

Produce a written project report. Your report should include the following:

1. Your name(s), CSUF-supplied email address(es), and an indication that the submission is for project 2.
2. The output of your exhaustive search algorithm for the following values of $n$ and $W$:

| $n$ | $W$ | Corresponding media |
|---|---|---|
| 20 | 1,000 | N/A |
| 23 | 720 | Low density 3.5" floppy disk |
| 23 | 1,440 | Standard 3.5" floppy disk |
| 23 | 25,600 | Maximum GMail attachment size |
| 23 | 65,536 | 64 MB flash thumb drive |

   Your output should include the elapsed running time, list of package names, and total size and total votes of the solution.
3. A scatter plot showing your code's run time as a function of the number of packages $n$. You can hold $W$ constant at $W = 2000$ while gathering this timing data.
4. Answers to the following questions, using complete sentences:
   (a) Is your algorithm's observed running time consistent with its predicted exponential efficiency class?
   (b) Find the largest value of $n$ that your exhaustive search algorithm can complete in 10 minutes, with $W = 2000$. What is this value?
   (c) Do you think this algorithm is fast enough to be used in practical applications? If so, under what conditions can it be used? Would it be a practical way for Debian to design their installation media?
   (d) Is this evidence *consistent* or *inconsistent* with the hypothesis stated on the first page? Justify your answer.
5. Your source code.

Your document *must* be uploaded to TITANium as a single PDF file.

## Due Date

The project deadline is Thursday, 10/9, 11:55 pm. Late submissions will not be accepted.