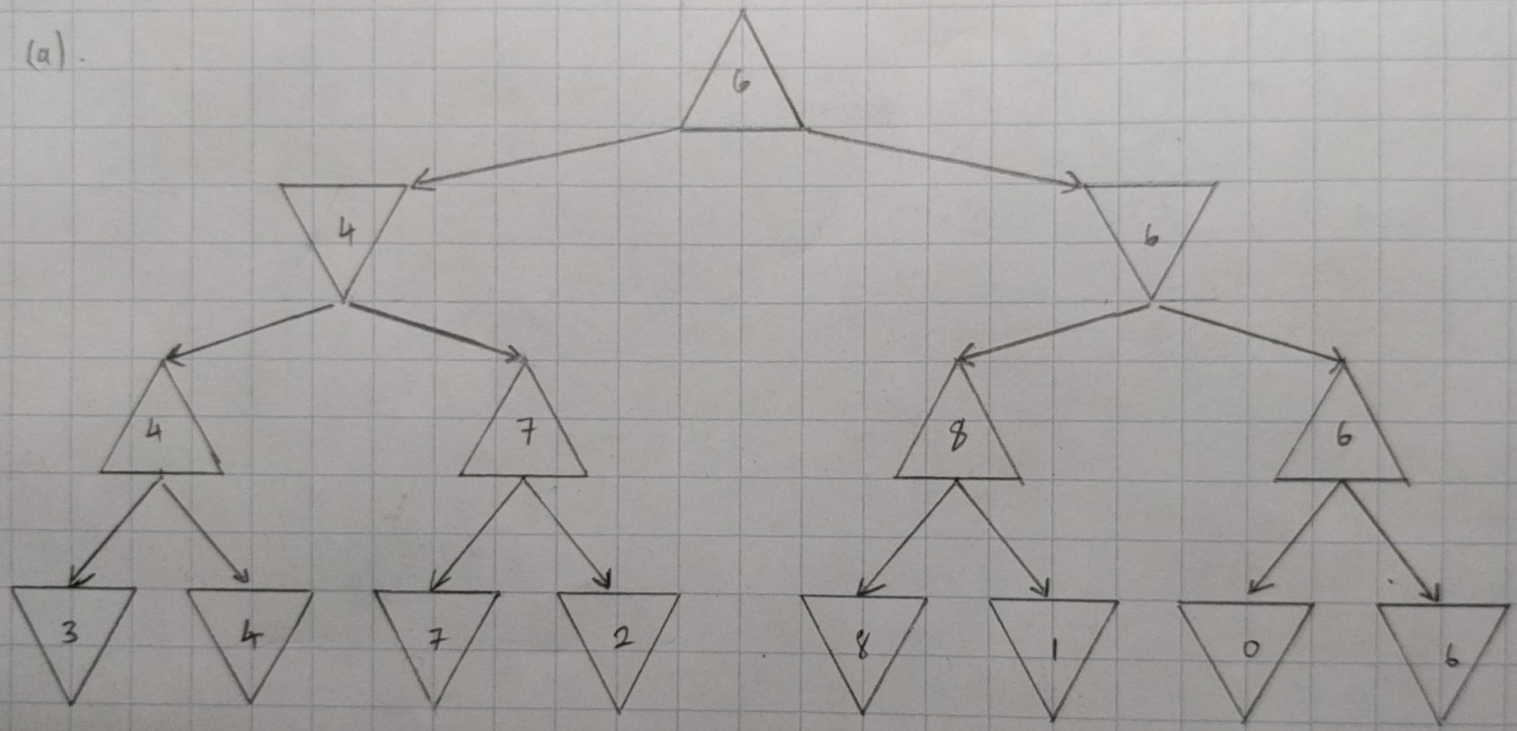
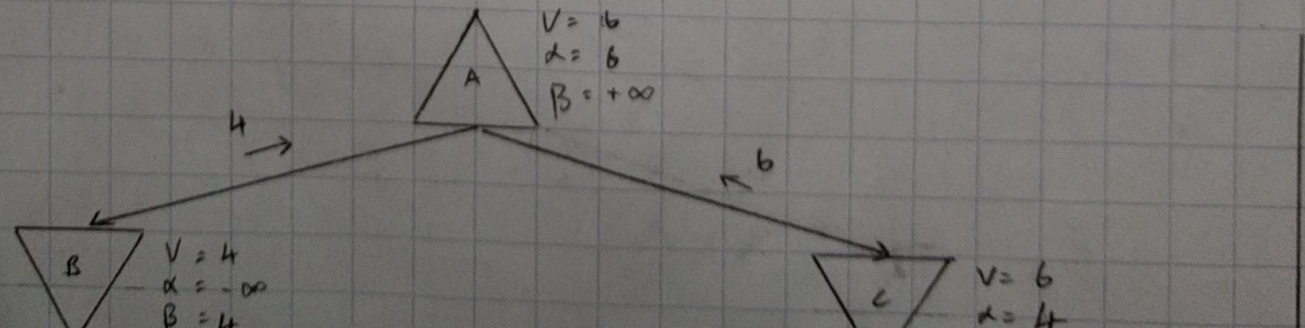
CS540 - HW5
Raunen Guha

Due: 03/12/18.

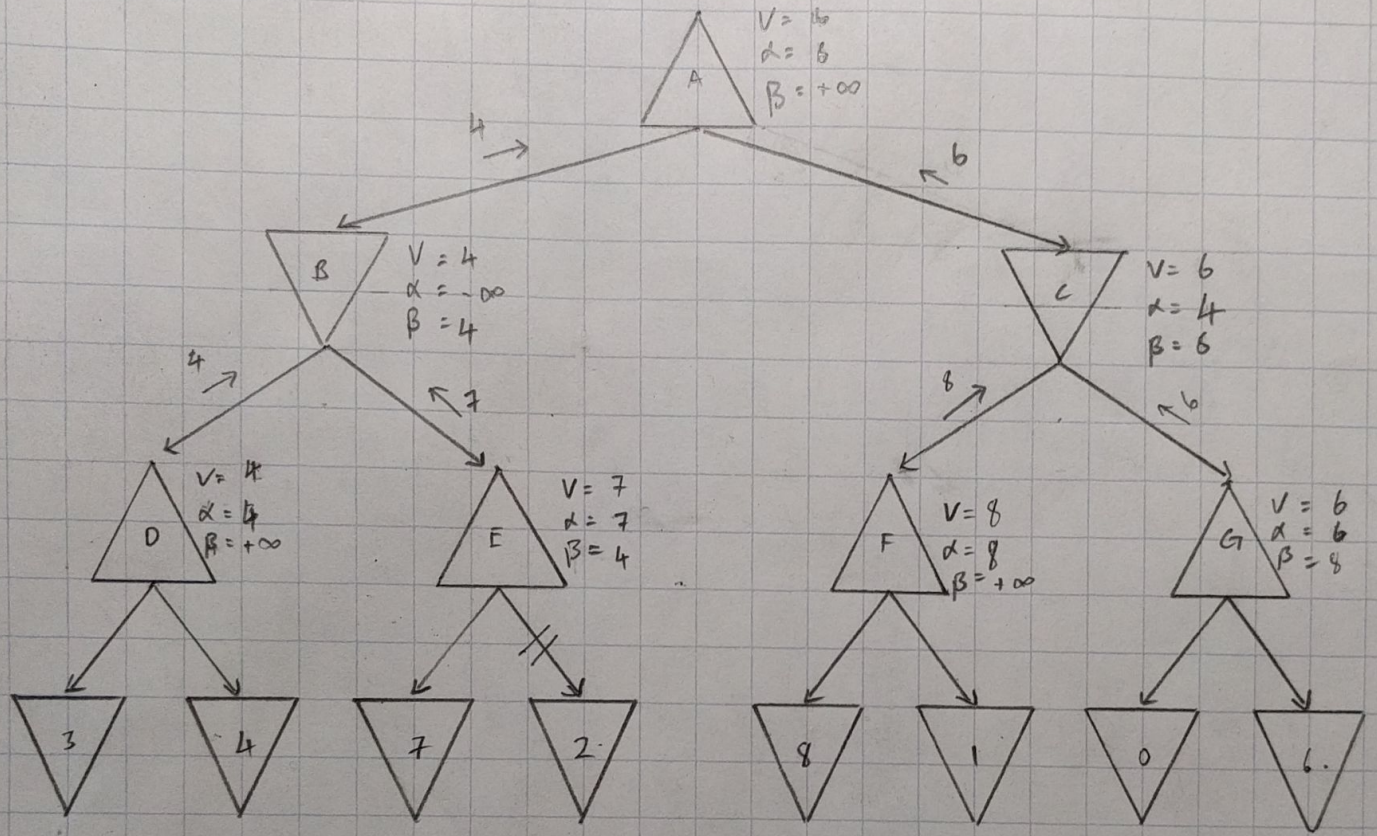
1. (a).




(b) $\alpha$: best already-explored option along path to the root for maximizer.
$\beta$: best already-explored option along path to the root for minimizer.



(b) $\alpha$: best already-explored option along path to the root for maximizer.
$\beta$: best already-explored option along path to the root for minimizer.




```
Max-Value(s, α, β):
    if terminal(s): return U(s);
    v = -∞;
    for c in next-states(s):
        v' = Min-Value(c, α, β);
        if v' > v: v = v';
        if v' ≥ β: return v;
        if v' > α: α = v';
    return v;
```

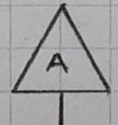
```
Min-Value(s, α, β):
    if terminal(s): return U(s);
    v = +∞;
    for c in next-states(s):
        v' = Max-Value(c, α, β);
        if v' < v: v = v';
        if v' ≤ α: return v;
        if v' < β: β = v';
    return v;
```

(c) The node with value 2 is pruned, as shown on the diagram.

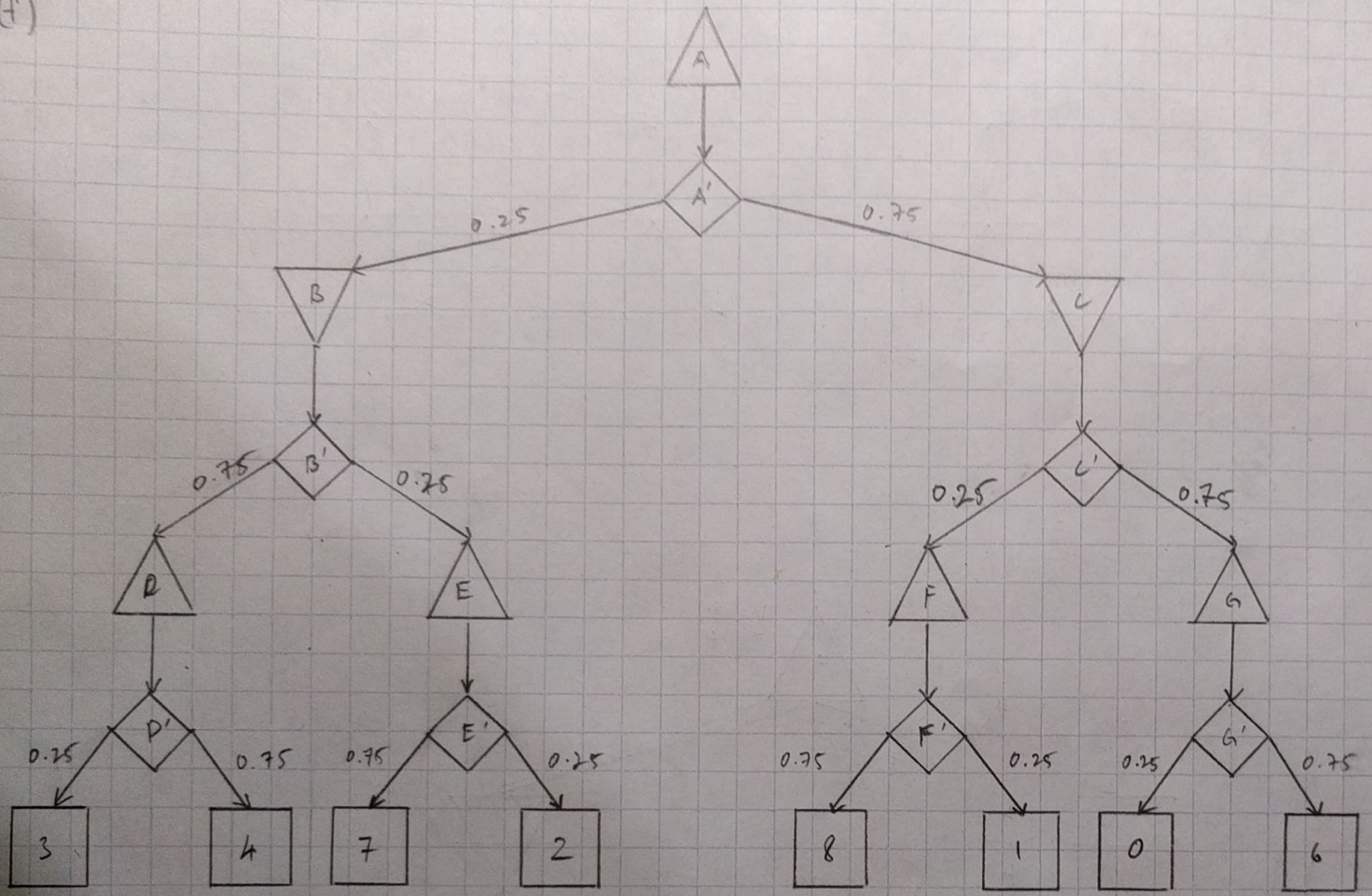
(d) Alpha-beta pruning is simply a more efficient implementation of Minimax, and in fact allows us to forego certain subtrees while still keeping the same final result

(e). $P(\text{optimal choice}) = P(\text{heads}) + P(\text{tails}) \cdot P(\text{optimal choice} \mid \text{tails})$
$= 0.5 + (0.5) \cdot (0.5)$
$= 0.75.$

(f)



(f)




(g).

$E(D) = 0.25(3) + 0.75(4)$
$= 3.75$

$E(E) = 0.75(7) + 0.25(2)$
$= 5.75$

$E(F) = 0.75(8) + 0.25(1)$
$= 6.25$

$E(G) = 0.75(6) + 0.25(0)$
$= 4.50$

$E(B) = 0.75(E(D)) + 0.25(E(E))$
$= 4.25$

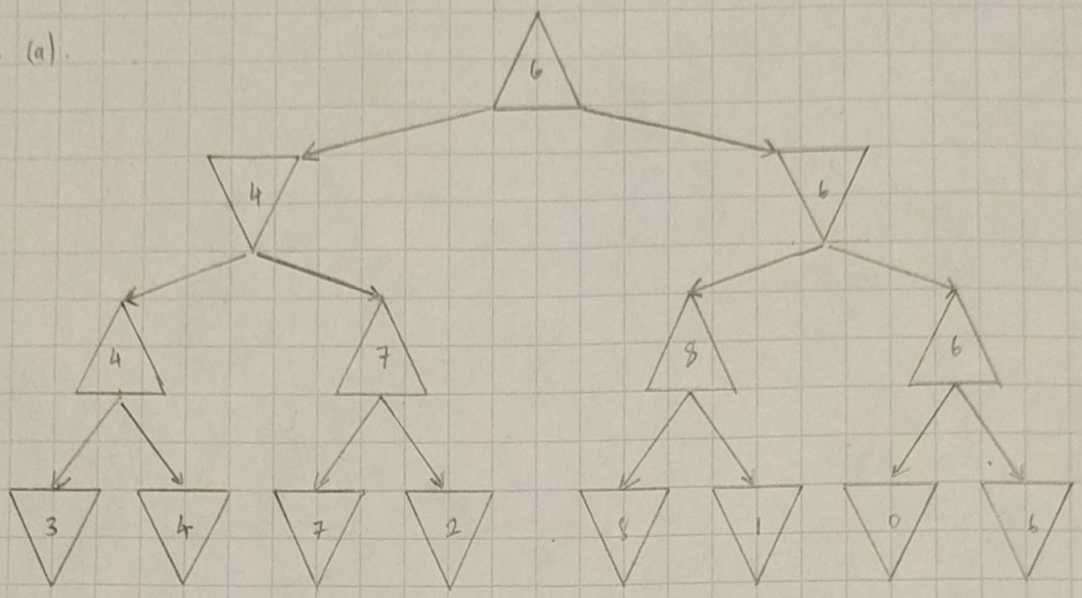
$E(C) = 0.25(E(F)) + 0.75(E(G))$.
$= 4.94$.

$E(A) = 0.25(E(B)) + 0.75(E(C))$
$= 4.77$

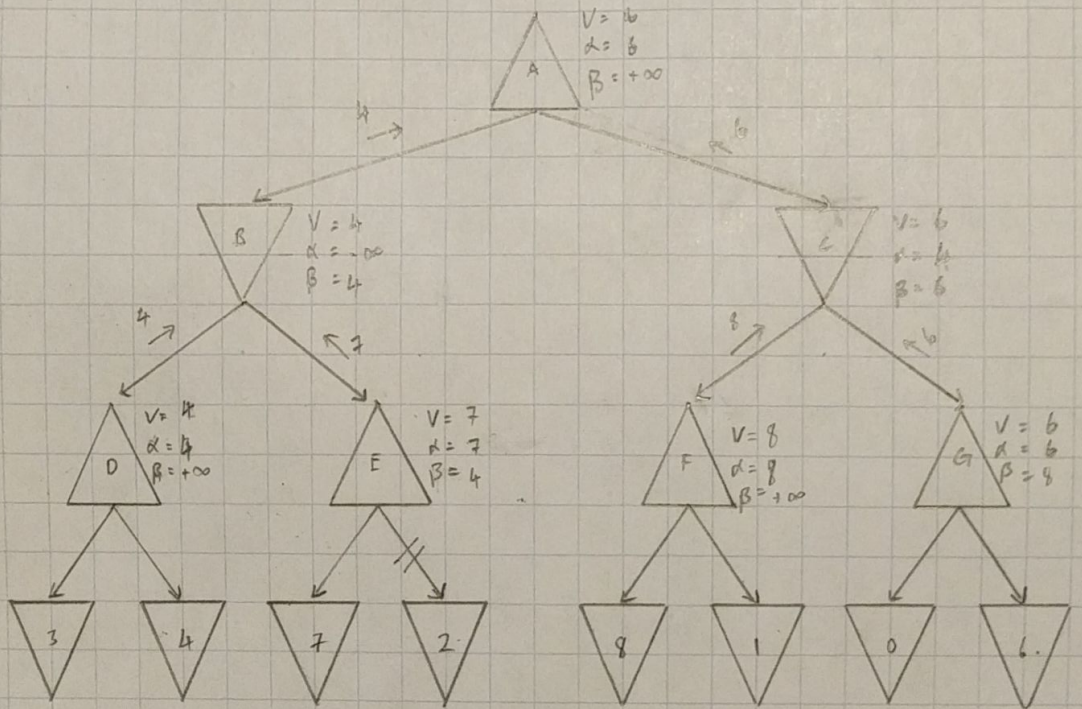
CS540 - HW5
Ramon Gruhn

Due: 03/12/18

1. (a).




(b) $\alpha$: best already-explored option along path to the root for maximizer.
$\beta$: best already-explored option along path to the root for minimizer.




```
Max-Value(s, α, β):
    if terminal(s): return U(s);
    v = -∞;
    for c in next-states(s):
        v' = Min-Value(c, α, β);
        if v' > v: v = v';
        if v' ≥ β: return v;
        if v' > α: α = v';
    return v;
```

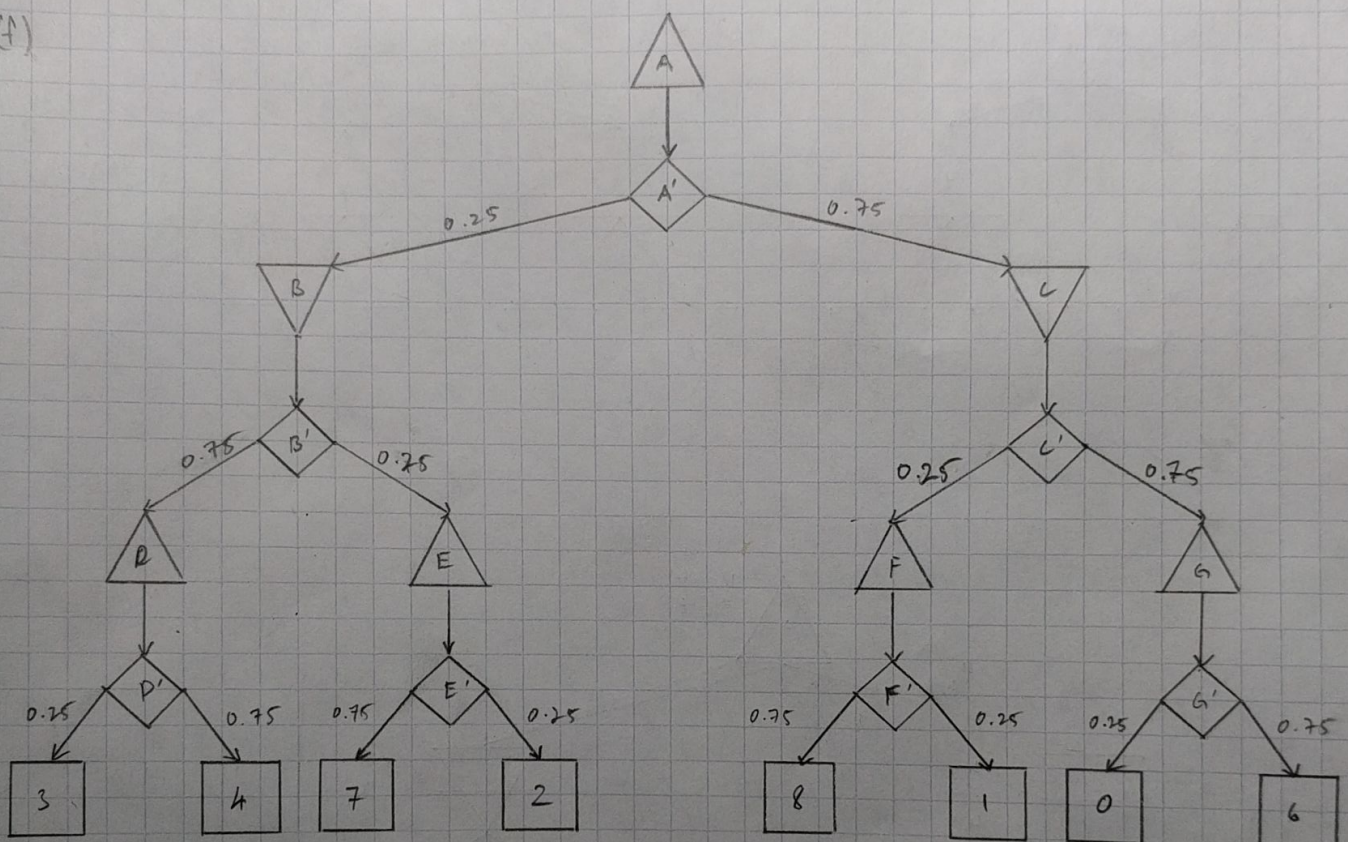
```
Min-Value(s, α, β):
    if terminal(s): return U(s);
    v = +∞;
    for c in next-states(s):
        v' = Max-Value(c, α, β);
        if v' < v: v = v';
        if v' ≤ α: return v;
        if v' < β: β = v';
    return v;
```

(c) The node with value 2 is pruned, as shown on the diagram.

(d) Alpha-beta pruning is simply a more efficient implementation of Minimax, and in fact allows us to forego certain subtrees while still keeping the same final result.

(e) P(optimal choice) = P(heads) + P(tails) · P(optimal choice | tails)
= 0.5 + (0.5) · (0.5)
= 0.75.

(f)




(g). E(D) = 0.25(3) + 0.75(4)
= 3.75

E(E) = 0.75(7) + 0.25(2)
= 5.75

E(F) = 0.75(8) + 0.25(1)
= 6.25

E(G) = 0.75(6) + 0.25(0)
= 4.50

E(B) = 0.75(E(D)) + 0.25(E(E))
= 4.25

E(C) = 0.25(E(F)) + 0.75(E(G)).
= 4.94.

E(A) = 0.25(E(B)) + 0.75(E(C))
= 4.77

## Table of Contents



# CS 540 - Spring 2018 - HW5

Question 2 by Roumen Guha

```
clc
clear
close all

addpath('C:\Users\roume\OneDrive\Documents\Classes\4 - Senior
\Spring 2018\CS 540 - Intro to Artificial Intelligence\Assignments
\HW5\WARC201709')
source_dir = 'C:\Users\roume\OneDrive\Documents\Classes\4 - Senior
\Spring 2018\CS 540 - Intro to Artificial Intelligence\Assignments
\HW5\WARC201709'; %uigetdir([]);

d = dir([source_dir, '\*.txt']);
n = length(d(not([d.isdir])));
```

# (a) Brief explanation of method

I implement MATLAB for this task (and live to regret it). We take in the strings from the filepath and preprocess them by first converting them to lowercase, then removing punctuation, then spliting them by "whitespace".

# (b)(i) Total words across n essays

```
total_words = 0;

for i = 1:n
    txt{i} =
 cellstr(strvcat(strsplit(erasePunctuation(lower(fileread(d(i).name))))))';
    total_words = total_words + numel(txt{i});
end

total_words
```

```
total_words =

      205029
```

# b(ii) Total distinct (unique) words across n essays

```
total_txt = cat(2, txt{:});
[total_distinct_txt, b, c] = unique(total_txt);
total_distinct_txt_count = hist(c, length(total_distinct_txt));

total_distinct_words = numel(total_distinct_txt);

total_distinct_words
```

```
total_distinct_words =

       11236
```

# (c) 20 most frequent words and their counts

```
[decreasing_val decreasing_ind] =
 sort(total_distinct_txt_count, 'descend');

total_distinct_txt{decreasing_ind(1:20)};
twenty_most_frequent_counts = decreasing_val(1:20);

for i = 1:20
    fprintf('%s: ', total_distinct_txt{decreasing_ind(i)});
    fprintf('%d \n', decreasing_val(i));
end

fprintf('\n\n');
```

```
the: 11816
to: 6782
of: 6152
and: 5252
in: 4411
a: 3892
that: 3431
is: 3373
ai: 3105
be: 2696
will: 2056
it: 1952
for: 1857
```

```
as: 1699
are: 1632
on: 1575
not: 1550
this: 1437
with: 1359
intelligence: 1181
```

# (d) 20 least frequent words and their counts

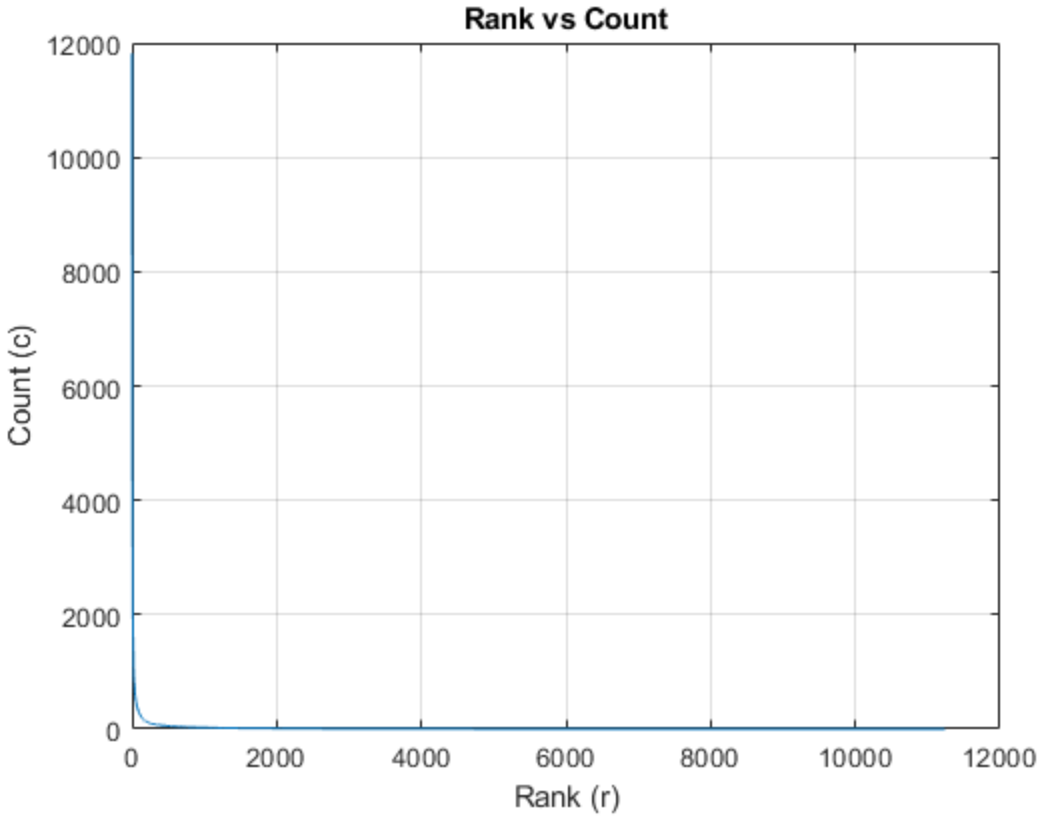
```
[increasing_val increasing_ind] =
 sort(total_distinct_txt_count, 'ascend');

total_distinct_txt{increasing_ind(1:20)};
twenty_least_frequent_counts = increasing_val(1:20);

for i = 1:20
    fprintf('%s: ', total_distinct_txt{increasing_ind(i)});
    fprintf('%d \n', increasing_val(i));
end

fprintf('\n\n');
```

```
01: 1
02: 1
03: 1
05: 1
09092017: 1
09132017: 1
10000: 1
100000: 1
1000mile: 1
101007s1067701697452: 1
101016japergo201509009: 1
1020: 1
104: 1
107: 1
111: 1
1119: 1
11717: 1
11th: 1
126: 1
12751289: 1
```

# (e) Plot rank r vs count c

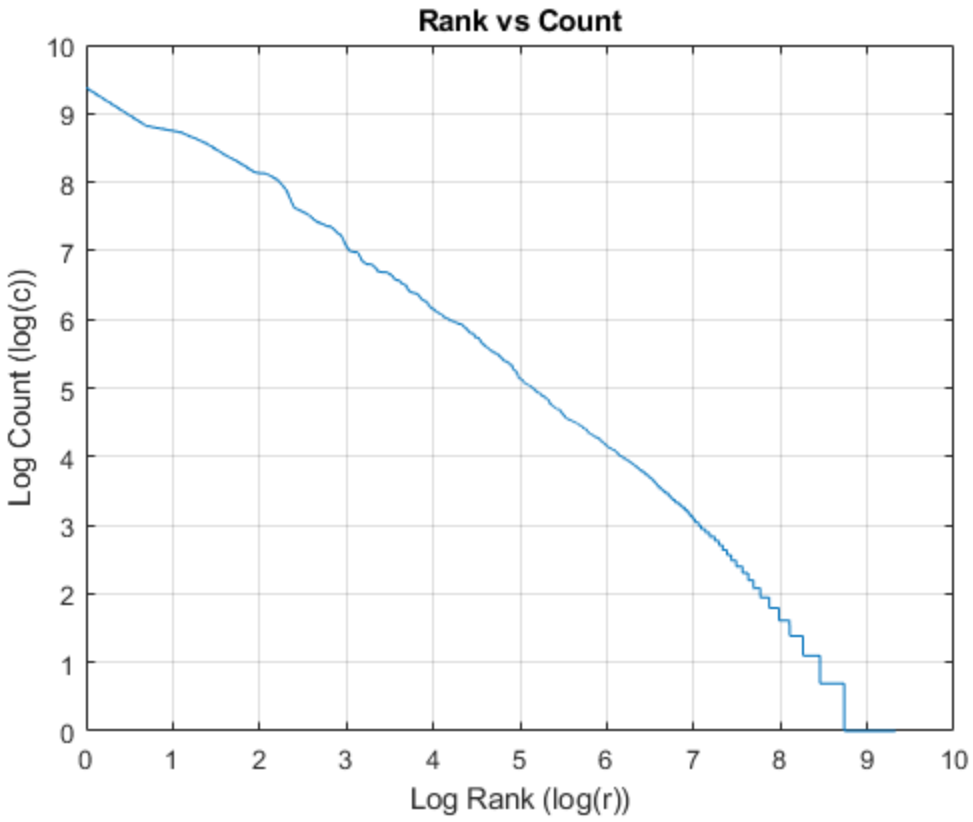
```
figure()
plot(decreasing_val)
```

```
grid
xlabel('Rank (r)');
ylabel('Count (c)');
title('Rank vs Count');
```




# (f) Plot log(r) vs log(c)

```
figure()
x = log(1:length(total_distinct_txt_count));
y = log(decreasing_val);
plot(x, y)
grid
xlabel('Log Rank (log(r))');
ylabel('Log Count (log(c))');
title('Rank vs Count');
```




# (g) Explain the shapes of the curves

The plot in part (e) shows the how much we humans rely heavily on the same words (at least in English). It is exponentially falling.

```
% The plot in part (f) shows the logarithmic plot, and it brings some
 credance
% to the fact that the previous plot is exponential, because this plot
 is almost
% linear.
```

# (h) Two potential issues with this method of preprocessing

I didn't account for misspellings, so those words were counted as unique words, instead of being corrected to words that were already in our list. It also did not count aliasing between words with equivalent meanings but different representations, such as between "eleventh" and "11th". Also, simply considering each number as a word would probably just add noise to our data, while giving us little to no new information, so they should probably be removed.

```
% Besides all this, there was also the issue of some instances of a
 word type
% being lost due to the typeset of my OS, so the counts of some words
 were also
```

```
% off.
```

*Published with MATLAB® R2017b*