Problem 1

i/v	s	a	b	c	t
1	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
2	0	1	2	$+\infty$	$+\infty$
3	0	1	2	4	3
4	0	1	2	4	2
5	0	1	2	4	2

Table 1. Bellman-ford Shortest Path

i/v	a	b	c	t
1	null	null	null	null
2	s	s	null	null
3	s	s	b	b
4	s	s	b	c
5	s	s	b	c

Table 2. Back Tracing of Previous Hop

From Table (1), we could do one more iteration loop for detecting cycle and since all columns are the same with the last iteration – 5, we can assure that there is no cycle added to the shortest path.

From Table (2), we can do backtracing from $T[5][t]$, the result c is the entry with which we should consult for previous iteration – 4.

Finally, the shortest path $s \to b \to c \to t$ is found.

Problem 3

Let $f(t, n)$ be the maximum value of m that is distinguishable using t tries and n eggs. The base case is that when we only have one trial left, given $n \geqslant 1$, no matter how many eggs we are left, we could only do one more trial; when we only have one egg left, at most we could do t more trials, where $t \geqslant 2$.

$$\begin{cases} f(1,n) = 1 \text{ for } n \geqslant 1 \\ f(t,0) = 0 \text{ for } t = 1 \\ f(t,1) = t \text{ for } t \geqslant 2 \end{cases}$$

Suppose floor x is from which given t eggs and n eggs in total we drop our first egg. For every drop, there are two possible scenarios:

i. The egg didn't get broken, so we could still do further trials on the upper floor with the same number of eggs (n) and with one less trial ($t-1$).

ii. The egg got broken, it means that critical floor m is between 0 to x. Also, we are only left with $t-1$ trials and $n-1$ eggs.

$f(t, n)$ should be as capable as covering the current x floors and its upper floor if current egg did't break, so we have:

$$f(t,n) = f(t-1,n) + x$$

To ensure scenario (ii) that we could still find out the critical floor m, in the Worst Case where every egg was dropped to be broken. So we have to ensure that $f(t-1, n-1)$ should be big enough to cover the bottom $x-1$ floors:

$$x - 1 \leqslant f(t-1, n-1)$$

We also want to be ambitious enough that the floor m could be found as soon as possible, so floor x should be chosen to be as maximum as possible. Now $f(t, n)$ would be:

$$f(t,n) = f(t-1,n) + f(t-1,n-1) + 1$$

Now let k be number of floors for a building such that given n eggs and t tries, we are assured to find out the critical floor m from which the egg must be dropped to be broken, under the Worst Case Scenario where every egg was dropped to be broken. The problem is turned into finding the minimum trial limitation s that satisfies $f(s, n) \geqslant k$. So we derive a dynamic programming approach for storing the intermediate results starting from $f(1, 1)$ increasing n and we will end with $f(\log(k), n)$ doing binary search on t to find s.

The time and space complexity are both $O(n \log(k))$.

Problem 4

1. Length of palindromic subsequence.

   a) a (1)

   b) bcacb (5)

   c) dbcacbd (7)

   d) sallas (6)

   e) aaaa (4)

2. Dynamic algorithm design for sub-palindrome problem.

   - Let T to be a $N \times N$ matrix, where N denotes the length of the input sequence $x_0x_1....x_{n-1}$. $T_{(i,j)}$ means that we pick a sub-string from input sequence which starts from position-i and ends at position-j, where i and j satisfy the condition $i \leqslant j$.

   - We will fill in the table from the diagonal line, which will be filled in from bottom-right to upper-left, to the upper-right corner.

   - $$T_{(i,j)} = \begin{cases} \max(T_{(i+1,j)}, T_{(i,j-1)}) & (\text{if } j - i \leqslant 1) \\ 2 + T_{(i+1,j-1)} & (\text{if } x_i = x_j) \\ j - i & (\text{otherwise}) \end{cases}$$

   - Pseudo-code:

```
sub_palindrome(string, i, j):
    if j - 1 <= 1:
        return 1
    if string[i] == string[j]:
        return 2 + sub_palindrome(string, i + 1, j - 1)
    else:
        return max(sub_palindrome(string, i + 1, j),
                   sub_palindrome(string, i, j - 1))
```

   - Correctness and complexity:

     a) Optimal Substructure

We want the result with the longest subsequence which is a palindrome. Let sequence $S = x_m x_{m+1}....x_n$ is the continguous subsequence with the maximum palindrome subsequence in it of the original sequence X, it could be narrowed down to 2 cases: (Case 1) x_m matched with x_n, the next optimal subsequence should be contained in $x_{m+1}x_{m+2}....x_{n-1}$; (Proof of Case 1) if S^* is better than S with regarding to the sequence $x_{m+1}x_{m+2}....x_{n-1}$, then S^* should be better than S with regarding to the sequence $x_m x_{m+1}....x_n$, which leads to contradiction with our assumption; (Case 2) otherwise, the next longest subsequence would be contained in either $x_{m+1}x_{m+2}....x_n$ or $x_m x_{m+1}....x_{n-1}$, and since every time we pick the one with the longest palindrome subsequence we could assure that we have the optimal subsequence in case 2; (Proof of Case 2) if the next longest subsequence is contained in $x_{m+1}x_{m+2}....x_n$, and S^* is better than S with regarding to the sequence $x_{m+1}x_{m+2}....x_n$, then S^* would equally be better than S with regarding to the sequence $x_m x_{m+1}....x_n$, contradicting with our original assumption with S. A similar proof could be conducted if the next longest subsequence is contained in $x_m x_{m+1}....x_{n-1}$.

Since every time when we narrow down the subsequence, we have assured that the longest palindromic subsequence had been taken into account, the overall computation is optimal.

b) Complexity

Since we need to fill in the table from bottom-right to top-left, and then till the upper-right corner, we only need at most half of the table, which is $\frac{n^2}{2}$. Therefore, the total time and space complexity are both $O(n^2)$.

Problem 5

1. Let m be the length of a line, n be the total number of words, and we label every word with indices in an increasing order from 1 to n.

   Let $P_{(i,j)}$ be the penalty value when we choose words from index i to index j. If current end, j, is not the last word, namely we are handling from the first line to the penultimate line, the penalty for each of those lines should be the maximum line capacity minus the spaces (assuming each pair of words is seperated by only one space at most), and then minus the total word length summed up, the rest then should be squared. If we are handling the last line and the last line could be filled into the m characters space, its penalty would be neglected according to the definition, otherwise, the case is not allowed giving a plus infinite number:

$$P_{(i,j)} = \begin{cases} (m-(j-i)\times 1-\sum_{k=i}^{j}\text{WordLen}(k))^2 & (j \neq n) \\ 0 & (j=n \text{ and } m-(j-i)\times 1-\sum_{k=i}^{j}\text{WordLen}(k) \geqslant 0) \\ +\infty & (j=n \text{ and } m-(j-i)\times 1-\sum_{k=i}^{j}\text{WordLen}(k) < 0) \end{cases}$$

   Let k be the current index of the last word chosen to be fit in the current line. Let $f(k)$ be the optimal alignment with minimum penalty using the first k words. So, if the current split could be tolorated by the current line, then the minimum penalty would just be $P_{(1,k)}$ itself. Otherwise, we should find an optimal way of splitting the line for every word-s from index 1 to k:

$$f(k) = \begin{cases} P_{(1,k)} & (m-(j-i)\times 1-\sum_{k=i}^{j}\text{WordLen}(k) \geqslant 0) \\ \min_{1 \leqslant s < k}\{f(s)+P_{(s+1,k)}\} & (\text{otherwise}) \end{cases}$$

2. Correctness

3. Time Complexity