Problem Set 11

Daniel Wang (S01435533)

1. The DFS is as below (underline boldface state is the end state):

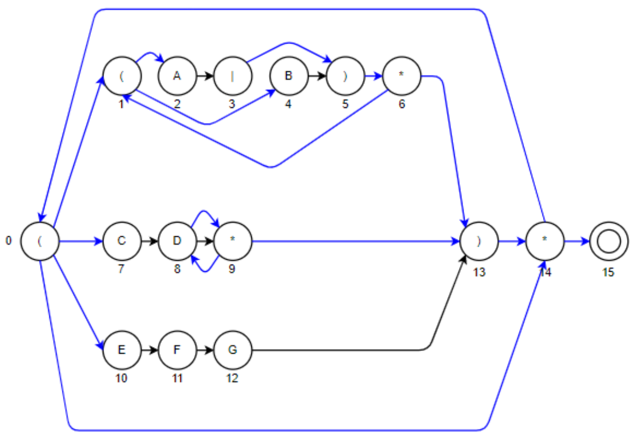
	0	1	2	3	4	5	6
pat	A	B	C	D	A	B	D
A	1	1	1	1	5	1	1
B	0	2	0	0	0	6	0
C	0	0	3	0	0	0	3
D	0	0	0	4	0	0	**7**

Using KMP algorithm, the state transition is as follows:

A	B	C	A	B	C	D	A	B	C	D	A	B	D
1	2	3	1	2	3	4	5	6	3	4	5	6	**7**

2. The answers are as follows:
   (1) Wildcard is equivalent to (a|b|c|d|e)
   (2) (RE)+ is equivalent to (RE)(RE)*
   (3) ab{3,5} is equivalent to abab(ab|ababab)
   (4) a[b-d] is equivalent to a(b|c|d)

3. The answers are as follows:
   (1) The NFA is shown as follows, where blue edges represent e-transitions.



(2) The parsed character with respect to possible states are as follows:

parser	Possible states
(start)	{0,1,2,4,7,10,14,15}
A	{0,1,2,3,4,7,10,14,15}
B	{0,1,2,4,5,7,10,14,15}

B	{0,1,2,4,5,7,10,14,15}
A	{0,1,2,3,4,7,10,14,15}
C	{0,1,2,4,7,8,10,14,15}
E	{11}
F	{12}
G	{0,1,2,4,7,10,13,14,15}
E	{11}
F	{12}
G	{0,1,2,4,7,10,13,14,15}
C	{0,1,2,4,7,8,10,14,15}
A	{0,1,2,3,4,7,10,14,15}
A	{0,1,2,3,4,7,10,14,15}
B	{0,1,2,4,5,7,10,14,15}
(end)	{15}

4. The Python code is shown as follows, where the algorithm is in linear time.

```python
def maximum_subarray(nums):
    """
    each loop:
    (1) keep track of previous minimum cumulative sum
    (2) update cumulative sum
    (3) keep track of current maximum cumulative sum
    """

    n = len(nums)
    max_sum = nums[0]
    min_sum = 0
    max_idx = min_idx = 0

    curr_sum = nums[0]
    for i in range(1, n):
        # update minsum from previous sum
        if curr_sum < min_sum:
            min_sum = curr_sum
            min_idx = i

        # update sum
        curr_sum += nums[i]

        # update maxsum from current sum
        if curr_sum > max_sum:
            max_sum = curr_sum
            max_idx = i

    return nums[min_idx : max_idx+1]
```

5. The answers are shown as follows:

   (1) The Python code is:

```python
def check_reconstitution(s):
    n = len(s)
    is_valid = [False for _ in range(n)]

    for i in range(n):
        for j in range(i):
            if is_valid[j] and dict(s[j+1:i+1]):
                is_valid[i] = True

        if dict(s[:i+1]):
            is_valid[i] = True

    return is_valid[n-1]
```

(2) The revised Python code is. The key is to keep track of previous valid indices.

```python
def get_reconstitution(s):
    if not check_reconstitution(s):
        return []

    n = len(s)
    source = [-1 for _ in range(n)] # previous source

    for i in range(n):
        for j in range(i):
            if source[j] != -1 and dict(s[j+1:i+1]):
                source[i] = j

        if dict(s[:i+1]):
            source[i] = i

    result = []
    curr_idx = n-1
    while curr_idx != source[curr_idx]:
        prev_idx = source[curr_idx]
        result.append(s[prev_idx+1:curr_idx+1])
        curr_idx = prev_idx

    result.append(s[:curr_idx+1])
    result.reverse()

    return result
```

6. The Python code is as follows:

```python
def minimum_matmul_cost(mat_sizes):

    def cols(i): return mat_sizes[i][1]
    def rows(i): return mat_sizes[i][0]

    n = len(mat_sizes)
    f = {}  # cached values of the recurrence relation

    for l in range(1, n + 1):
        for start in range(0, n - l + 1):
            # Base case
            if l == 1:
                f[(start, start)] = 0
```

```python
                continue

            # Recursive case
            end = start + l - 1
            f[(start, end)] = min(
                f[(start, mid)] +
                f[(mid + 1, end)] +
                rows(start) * cols(mid) * cols(end)
                for mid in range(start, end)  # end is exclusive
            )

    return f[(0, n - 1)]
```

7. The Python code is as follows:

```python
def minimum_edit_distance(s, t):
    m, n = len(s), len(t)
    min_costs = [[0 for _ in range(n+1)] for _ in range(m+1)]

    for i in range(m+1):
        min_costs[i][0] = i
    for i in range(n+1):
        min_costs[0][i] = i

    for i in range(1, m+1):
        for j in range(1, n+1):
            candidates = [min_costs[i-1][j]+1, min_costs[i][j-1]+1]
            if s[i-1] == t[j-1]:
                candidates.append(min_costs[i-1][j-1])
            else:
                candidates.append(min_costs[i-1][j-1]+1)
            min_costs[i][j] = min(candidates)

    return min_costs[m-1][n-1]
```