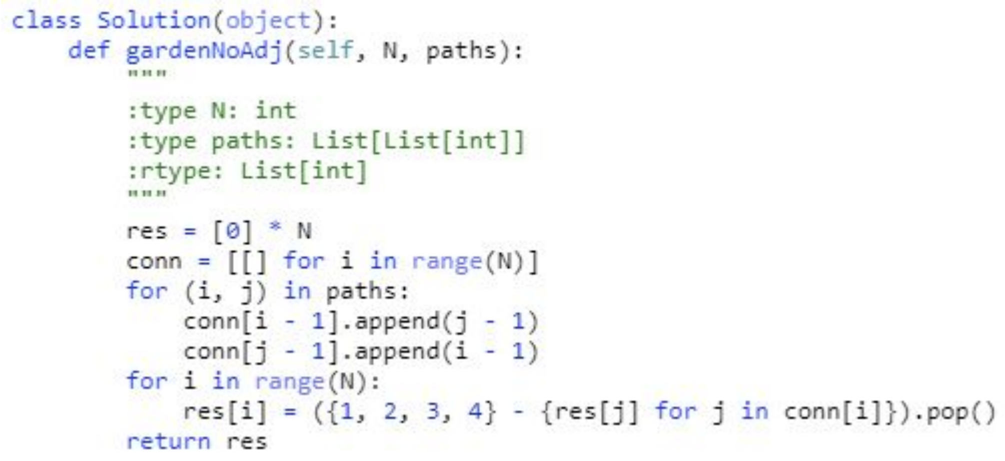
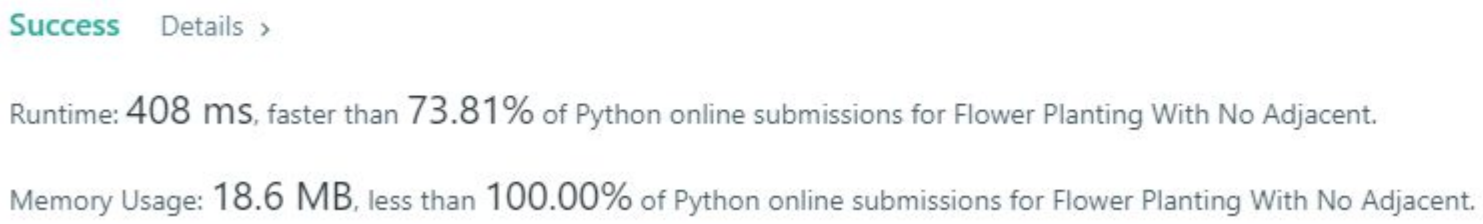
LC 1042: Flower Planting With No Adjacent



```python
class Solution(object):
    def gardenNoAdj(self, N, paths):
        """
        :type N: int
        :type paths: List[List[int]]
        :rtype: List[int]
        """
        res = [0] * N
        conn = [[] for i in range(N)]
        for (i, j) in paths:
            conn[i - 1].append(j - 1)
            conn[j - 1].append(i - 1)
        for i in range(N):
            res[i] = ({1, 2, 3, 4} - {res[j] for j in conn[i]}).pop()
        return res
```

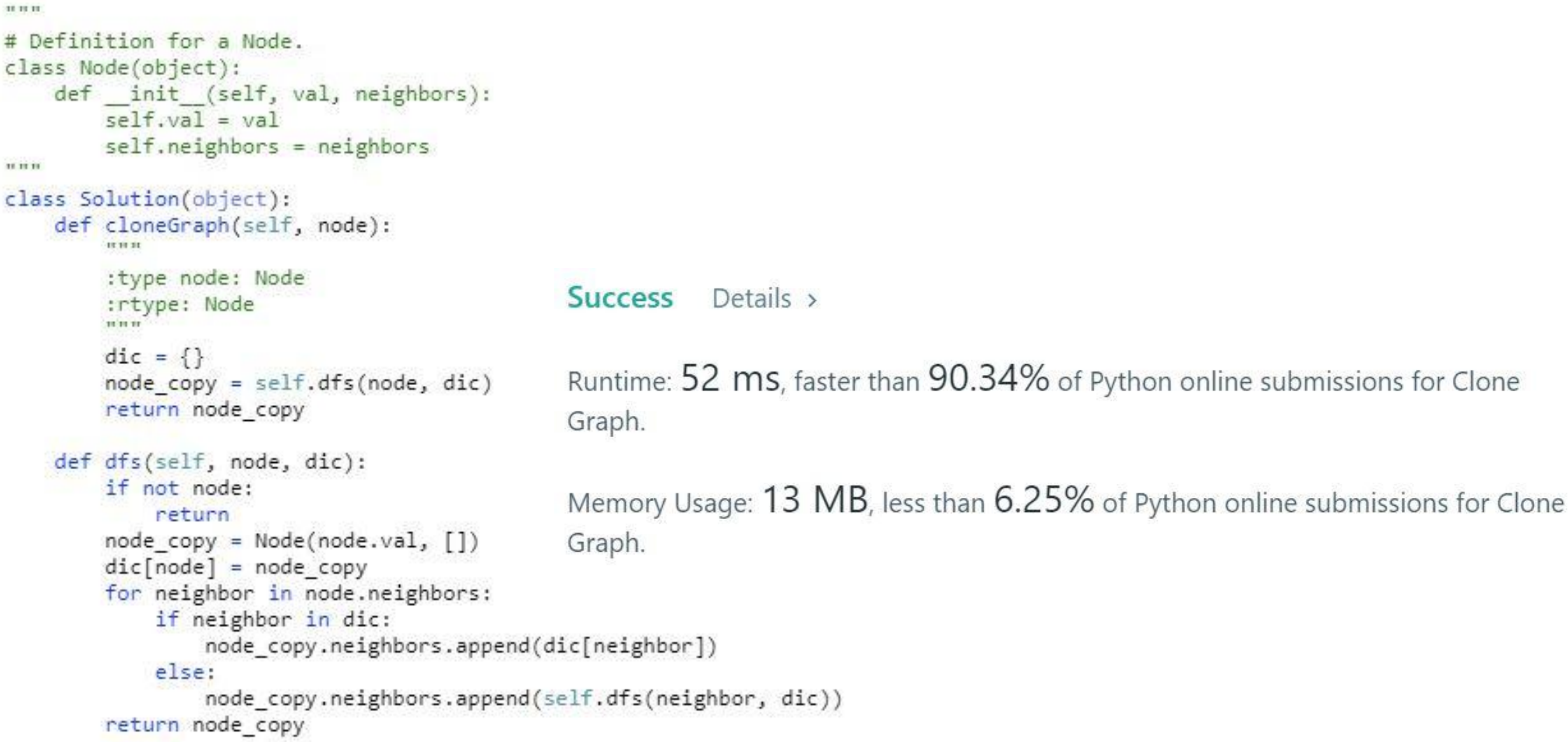


Success Details ›

Runtime: 408 ms, faster than 73.81% of Python online submissions for Flower Planting With No Adjacent.

Memory Usage: 18.6 MB, less than 100.00% of Python online submissions for Flower Planting With No Adjacent.

LC 133: Clone Graph



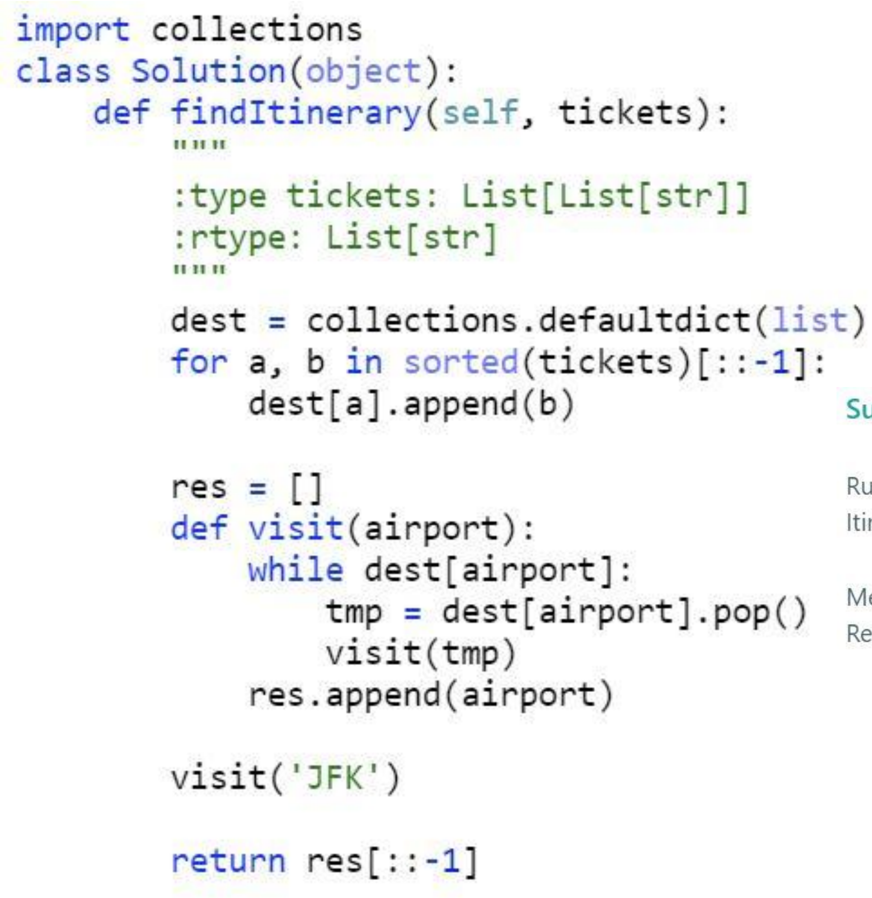
```
"""
# Definition for a Node.
class Node(object):
    def __init__(self, val, neighbors):
        self.val = val
        self.neighbors = neighbors
"""
class Solution(object):
    def cloneGraph(self, node):
        """
        :type node: Node
        :rtype: Node
        """
        dic = {}
        node_copy = self.dfs(node, dic)
        return node_copy

    def dfs(self, node, dic):
        if not node:
            return
        node_copy = Node(node.val, [])
        dic[node] = node_copy
        for neighbor in node.neighbors:
            if neighbor in dic:
                node_copy.neighbors.append(dic[neighbor])
            else:
                node_copy.neighbors.append(self.dfs(neighbor, dic))
        return node_copy
```

Success

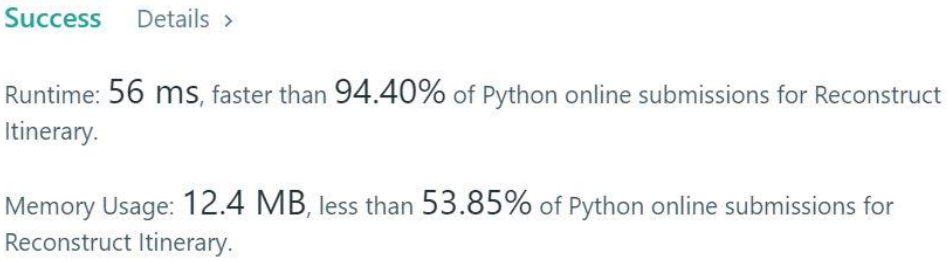
Runtime: 52 ms, faster than 90.34% of Python online submissions for Clone Graph.

Memory Usage: 13 MB, less than 6.25% of Python online submissions for Clone Graph.

LC 332: Reconstruct Itinerary



```
import collections
class Solution(object):
    def findItinerary(self, tickets):
        """
        :type tickets: List[List[str]]
        :rtype: List[str]
        """
        dest = collections.defaultdict(list)
        for a, b in sorted(tickets)[::-1]:
            dest[a].append(b)

        res = []
        def visit(airport):
            while dest[airport]:
                tmp = dest[airport].pop()
                visit(tmp)
            res.append(airport)

        visit('JFK')

        return res[::-1]
```

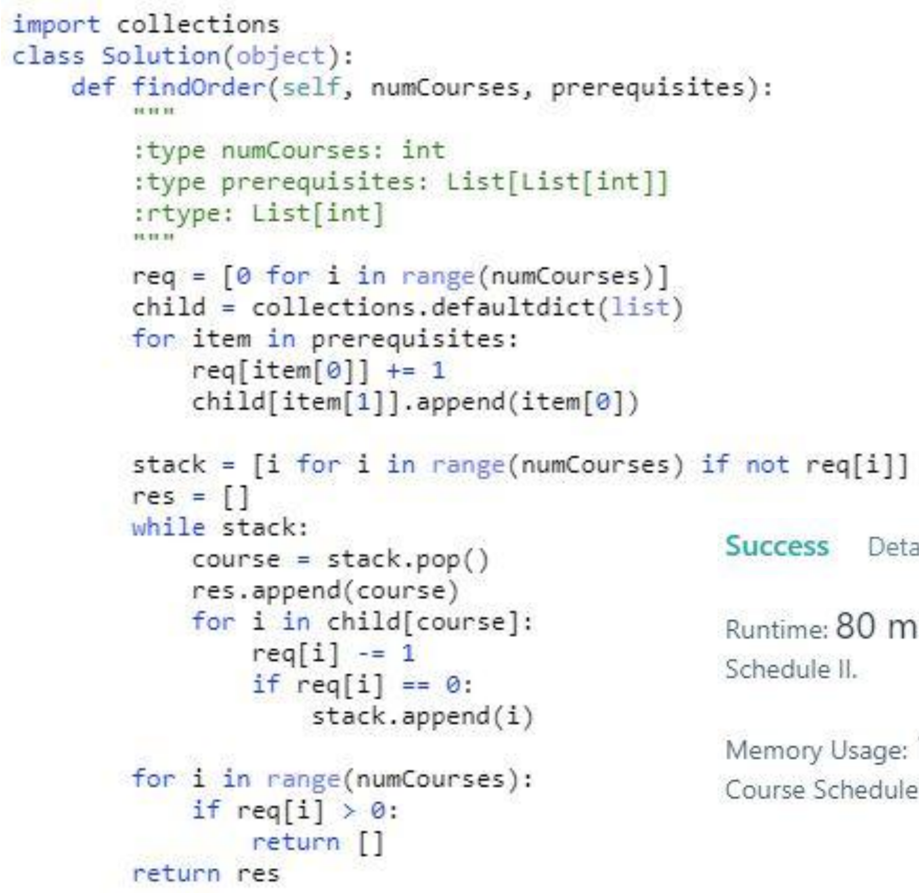


Success Details ›

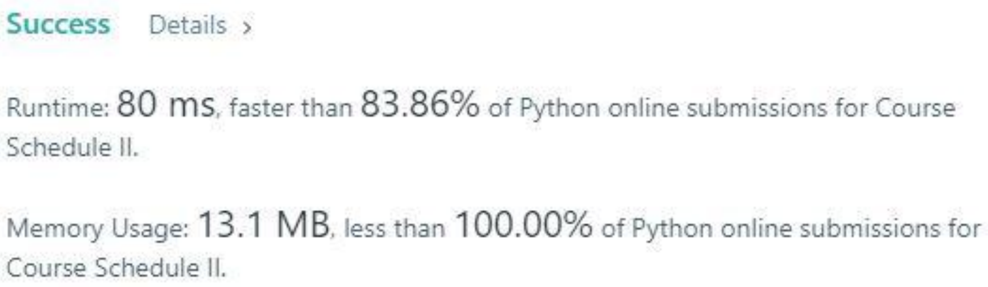
Runtime: 56 ms, faster than 94.40% of Python online submissions for Reconstruct Itinerary.

Memory Usage: 12.4 MB, less than 53.85% of Python online submissions for Reconstruct Itinerary.

LC 210: Course Schedule II



```python
import collections
class Solution(object):
    def findOrder(self, numCourses, prerequisites):
        """
        :type numCourses: int
        :type prerequisites: List[List[int]]
        :rtype: List[int]
        """
        req = [0 for i in range(numCourses)]
        child = collections.defaultdict(list)
        for item in prerequisites:
            req[item[0]] += 1
            child[item[1]].append(item[0])

        stack = [i for i in range(numCourses) if not req[i]]
        res = []
        while stack:
            course = stack.pop()
            res.append(course)
            for i in child[course]:
                req[i] -= 1
                if req[i] == 0:
                    stack.append(i)

        for i in range(numCourses):
            if req[i] > 0:
                return []
        return res
```

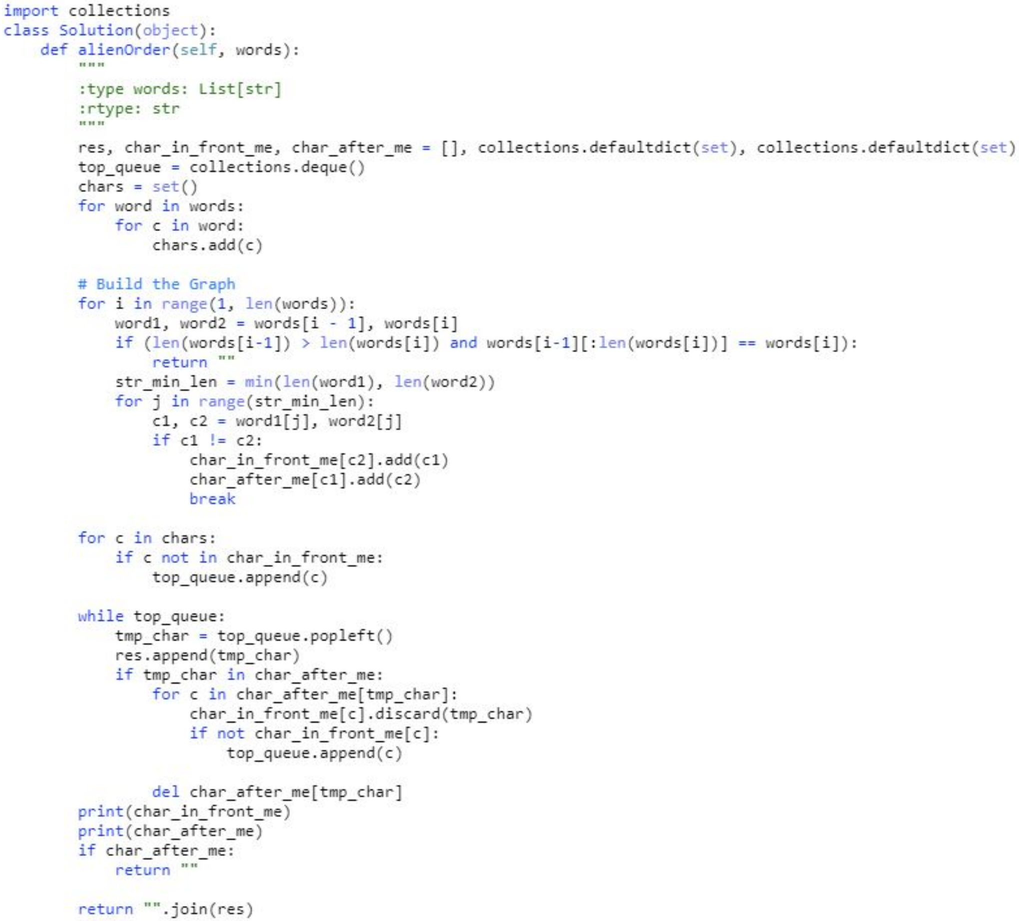


Success Details ›

Runtime: 80 ms, faster than 83.86% of Python online submissions for Course Schedule II.

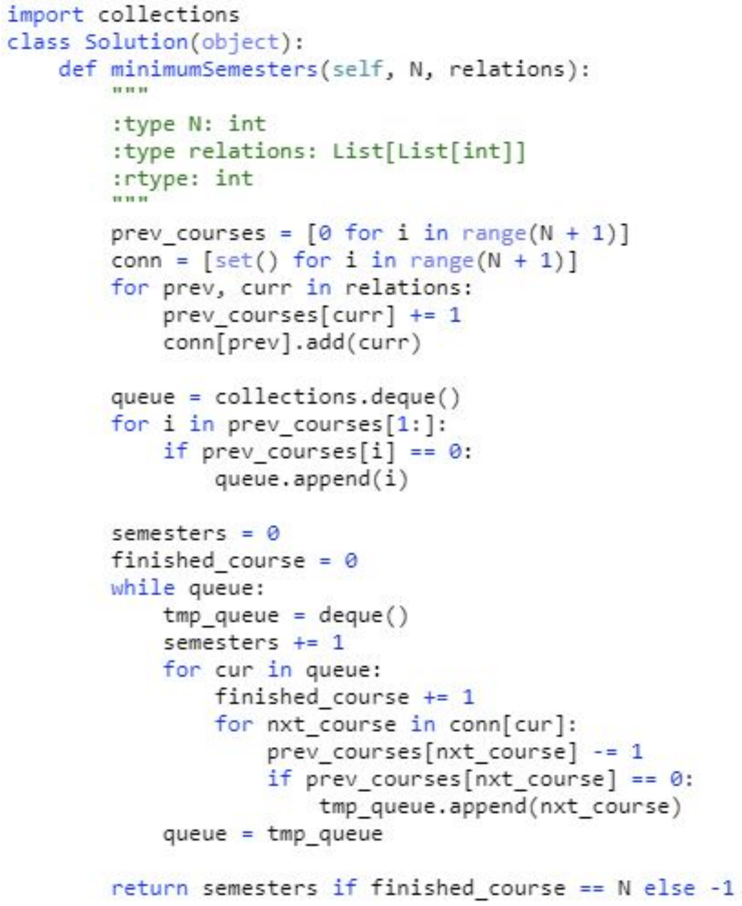
Memory Usage: 13.1 MB, less than 100.00% of Python online submissions for Course Schedule II.

LC 269: Alien Dictionary



```
import collections
class Solution(object):
    def alienOrder(self, words):
        """
        :type words: List[str]
        :rtype: str
        """
        res, char_in_front_me, char_after_me = [], collections.defaultdict(set), collections.defaultdict(set)
        top_queue = collections.deque()
        chars = set()
        for word in words:
            for c in word:
                chars.add(c)

        # Build the Graph
        for i in range(1, len(words)):
            word1, word2 = words[i - 1], words[i]
            if (len(words[i-1]) > len(words[i]) and words[i-1][:len(words[i])] == words[i]):
                return ""
            str_min_len = min(len(word1), len(word2))
            for j in range(str_min_len):
                c1, c2 = word1[j], word2[j]
                if c1 != c2:
                    char_in_front_me[c2].add(c1)
                    char_after_me[c1].add(c2)
                    break

        for c in chars:
            if c not in char_in_front_me:
                top_queue.append(c)

        while top_queue:
            tmp_char = top_queue.popleft()
            res.append(tmp_char)
            if tmp_char in char_after_me:
                for c in char_after_me[tmp_char]:
                    char_in_front_me[c].discard(tmp_char)
                    if not char_in_front_me[c]:
                        top_queue.append(c)

                del char_after_me[tmp_char]
        print(char_in_front_me)
        print(char_after_me)
        if char_after_me:
            return ""

        return "".join(res)
```

LC 1136: Parallel Courses



```python
import collections
class Solution(object):
    def minimumSemesters(self, N, relations):
        """
        :type N: int
        :type relations: List[List[int]]
        :rtype: int
        """
        prev_courses = [0 for i in range(N + 1)]
        conn = [set() for i in range(N + 1)]
        for prev, curr in relations:
            prev_courses[curr] += 1
            conn[prev].add(curr)

        queue = collections.deque()
        for i in prev_courses[1:]:
            if prev_courses[i] == 0:
                queue.append(i)

        semesters = 0
        finished_course = 0
        while queue:
            tmp_queue = deque()
            semesters += 1
            for cur in queue:
                finished_course += 1
                for nxt_course in conn[cur]:
                    prev_courses[nxt_course] -= 1
                    if prev_courses[nxt_course] == 0:
                        tmp_queue.append(nxt_course)
            queue = tmp_queue

        return semesters if finished_course == N else -1
```