Список вопросов на полузачёте

Алгоритмические и теоретические вопросы:

1. Понятия теории графов. Вершины, рёбра, петли, кратные рёбра, инцидентность, смежные вершины.
2. Понятия теории графов. Пути (цепи) и циклы в графах.
3. Понятия теории графов. Связность графов. Компоненты связности.
4. Понятия теории графов. Взвешенный граф.
5. Способы представления графа в памяти.
6. Сильно связные компоненты орграфов и ациклические орграфы. Алгоритм Косарайю (описание).
7. Три определения дерева. Остовное дерево графа.
8. Эйлеров цикл и Эйлеров путь. Эйлеровы, полуэйлеровы и не Эйлеровы графы. Привести примеры.
9. Гамильтонов цикл и Гамильтонов путь. Гамильтонов и полугамильтонов граф. Гамильтоновы, полугамильтоновы и не Гамильтоновы графы. Привести примеры.
10. Задача о китайском почтальоне. Задача о коммивояжёре.

Реализация алгоритма на языке Python 3:

1. Считывание матрицы смежности орграфа и вывод списков смежности.
2. Подсчёт компонент связности поиском в глубину. Реализация на Python 3.
3. Подсчёт компонент связности поиском в ширину. Реализация на Python 3.
4. Остовное дерево поиска в глубину. Реализация на Python 3.
5. Остовное дерево поиска в ширину. Реализация на Python 3.
6. Алгоритм Дейкстры с восстановлением кратчайшего пути. Реализация на Python 3.
7. Алгоритм Флойда-Уоршелла. Реализация на Python 3.
8. Алгоритм Прима. Реализация на Python 3.
9. Алгоритм Краскала. Реализация на Python 3.
10. Топологическая сортировка. Алгоритм Кана и Тарьяна. Реализация одного из них на Python 3.

Алгоритмические и теоретические вопросы

1. Граф — множество **вершин** и инцедентных им **ребер**.

$$G = (V, E)$$

$$v \in V, \qquad e \in E$$

Говорят, что ребро e **инцедентно** вершине v, если она является его концом.
Допустимы графы:

$$G = (\varnothing, \varnothing)$$

$$G = (1, \varnothing)$$

Недопустим граф:

$$G = (\varnothing, a)$$

Граф — "упрощенная модель".
У ребра <u>2 конца</u>. Это не обязательно отрезок.
Ребро может быть **петлей**.
2 разных ребра могут быть инцедентно двум вершинам — **кратные ребра**.
Смежные вершины — «соседи», т.е. это вершины, которые имеют общее ребро.
У классического графа 2 конца. Но может быть ориентированный граф. Т.е. либо у ребра 2 конца, либо у него есть начало и конец. Тогда ребро называется дуга. Короткое название **орграф**.

2. **Путь** — последовательность ребер (в которой конец каждого ребра есть начало следующего).
Путь тоже является графом, а точнее это орграф, подграф исходного.
Любой неориентированный граф можно представить как ориентированный.
Цикл — путь, в котором начало пути (начало первого ребра) совпадает с концом (конец последнего ребра).
Рассмотрим граф A—B.
Возможны пути

$$[AB, BA]$$

Простой путь — путь, у которого не повторяются ребра (вершины повторятся могут).
Простой цикл — цикл, у которого не повторяются ребра (вершины повторятся могут).
Эйлеров цикл — простой цикл, включающий все ребра графа.
3. Граф является **связным**, если для $\forall A{,}B \in V$ существует путь от A к B.
$A \longrightarrow B \longrightarrow C$ — несвязный граф.
Компонента связности — связный подграф, в который включены все вершины исходного, связаные с принадлежащими подграфами. Связный граф имеет 1 компоненту связности. Крайний случай: вершины без ребер. Количество компонент связности от 1 до количества вершин.
Слабая связность графа — "забываем" про направленность графов и смотрим на связность.
Сильно связный граф — граф связан при условии направленности.
4. **"Вес" ребра** — некоторая числовая характеристика ребра (расстояние, время прохождения, стоимость, энергия реакции и т.д.). Это необязательно положительное число.
Взвешенный граф — граф, у которого все ребра имеют вес.
5. Есть 3 основные формы хранения:
1. **Список ребер (множество ребер)**

AB 5
BC 3
CD 1
DE 2

2. **Матрица смежности**
Матрица смежности не умеет хранить кратные ребра (если только массив не трехмерный (но это бред)).

	A	B	C	D	E
A	x	1	0	0	0
B	1	x	1	0	0
C	0	1	x	1	0
D	0	0	1	x	1
E	0	0	0	1	x

Можно также составить матрицу взвешенности, если записать в эту матрицу вес каждого ребра.
3. **Списки смежности**

$$\begin{aligned} &A : B \\ &B : A, \quad C \\ &C : B, \quad D \\ &D : C, \quad E \\ &E : D \end{aligned}$$

6. **Ациклический граф** — орграф без цикла.
   Сильно связный граф — граф связан при условии направленности.
   Алгоритм Косарайю — алгоритм поиска компонент сильной связности в орграфе.
   Алгоритм:
   (a) Инвертируем ребра исходного орграфа.
   (b) Запускаем поиск в глубину на этом обращенном графе. В результате получаем вектор обхода.
   (c) Запускаем поиск в глубину на исходном графе, в очередной раз выбирая непосещённую вершину с максимальным номером в векторе, полученном в п.2.
   (d) Полученные из п.3 деревья и являются сильно связными компонентами.
7. **Дерево** — связный граф, в котором
   (a) Нет простых циклов
   (b) От a к b только один путь
   (c) $N_{\text{вершин}} = M_{\text{ребер}} + 1$

   Остовное дерево — пограф исходного графа, в котором выброшено максимальное количество ребер так, чтобы связность еще сохранилась. Обход графа в глубину позволяет построить одно из остовных деревьев.
8. **Эйлеров цикл** — простой цикл, включающий все ребра графа.
   Эйлеров путь (эйлерова цепь) в графе — это путь, проходящий по всем рёбрам графа и притом только по одному разу.
   Эйлеров граф — граф, в котором существует Эйлеров цикл.
   Полуйэлеров граф — граф, в котором есть Эйлеров путь, но нет Эйлерого цикла.
   Не эйлеров граф — граф, в котором не существует Эйлерова цикла.
   Для того, чтобы граф $G = (V, E)$ был эйлеровым необходимо чтобы:
   1. Все вершины имели четную степень.
   2. Все компоненты связности кроме, может быть одной, не содержали ребер.
   Примеры:
   Эйлеров граф: шестичленный цикл.
   Не эйлеров граф: квадрат с диагональю.

9. **Гамильтонов цикл** — цикл, проходящий через все вершины по одному разу.
   Гамильтонов путь — путь, проходящий через все вершины по одному разу.
   Граф называется **полугамильтоновым**, если он содержит гамильтонов путь.
   Граф называется **гамильтоновым**, если он содержит гамильтонов цикл.
   Граф называется **не гамильтоновым**, если он не содержит гамильтонов цикл.
   Пример гамильтонова цикла: пятиугольник.
10. **Задача о коммивояжере** — задача, в которой коммивояжер должен посетить N городов, побывав в каждом из них ровно по одному разу и завершив путешествие в том городе, с которого он начал. В какой последовательности ему нужно обходить города, чтобы общая длина его пути была наименьшей?
    Задача о коммивояжере относится к классу **NP-полных задач** (задачи, решаемые полным перебором).
    Задача о китайском почтальоне — задача, в котором почтальону нужно пройти по каждому ребру графа минимум 1 раз, чтобы доставить почту в каждую вершину. Требуется найти такой цикл минимального суммарного веса.
    Если граф эйлеров, то **эйлеров цикл является решением**. **Если есть вершины с нечетными степенями** (критерий эйлеровости графа), то решение находится **только полным перебором**.

1. Считывание графа как матрицы и как списка смежностей

```python
def read_graph_as_matrix():
    N, M = [int(x) for x in input().split()]
    graph = [[0]*N for i in range(N)] # матрица смежностей
    for edge in range(M):
        a, b = [int(x) for x in input().split()]
        graph[a][b] = 1
        graph[b][a] = 1
    return graph

def print2d(A):
    for line in A:
        print(*line)
    print()

def read_graph_as_lists():
    N, M = [int(x) for x in input().split()]
    graph = [[] for i in range(N)]
    for edge in range(M):
        a, b = [int(x) for x in input().split()]
        graph[a].append(b)
        graph[b].append(a) # Для ориентированного графа строка не нужна
    return graph

graph = read_graph_as_lists()
print2d(graph)
```

2. Реализация алгоритма обхода графа в глубину и подсчет компонент связности

```python
def dfs(vertex, graph, used = None): # Depth-first search
    if used is None:
        used = set()
    used.add(vertex)
    for neighbour in graph[vertex]:
        if neighbour not in used:
            dfs(neighbour, graph, used)

graph = read_graph_as_lists()
used = set()
number_of_components = 0
for vertex in range(len(graph)): # Подсчет компонент связности
    if vertex not in used:
        dfs(vertex, graph, used)
        number_of_components += 1

print('Количество компонент связности:', number_of_components)
```

3. **Реализация алгоритма BFS и подсчет компонент связности**

```
1  def bfs_fire(G, start, fired = None):
2      if fired is None:
3          fired = set()
4      fired.add(start)
5      time = {start: 0} # Хранение времен их добывания
6      Q = [start]
7      while Q:
8          current = Q.pop(0) # Для списка это не эффективно
9          for neighbour in G[current]:
10             if neighbour not in fired:
11                 fired.add(neighbour)
12                 Q.append(neighbour)
13                 print(current, neighbour) # Для построения остовного дерева
14                 time[neighbour] = time[current] + 1
15
16 graph = read_graph_as_lists()
17 used = set()
18 number_of_components = 0
19 for vertex in range(len(graph)): # Подсчет компонент связности
20     if vertex not in used:
21         bfs_fire(graph, vertex, used)
22         number_of_components += 1
23
24 print('Количество компонент связности:', number_of_components)
```

4. **Реализация алгоритма обхода графа в глубину и подсчет компонент связности**

```
def dfs(vertex, graph, used = None, tree = []): # Добавим tree
    if used is None:
        used = set()
    used.add(vertex)
    for neighbour in graph[vertex]:
        if neighbour not in used:
            tree.append((vertex, neighbour)) # К дереву добавим
            dfs(neighbour, graph, used, tree)
    return tree
```

5. Как в п.3. строка 13.

6. **Реализация алгоритма Дейкстры**

```
def dijkstra(G, start): # G - словарь словарей с весами
    d = {v: float('+inf') for v in G}
    d[start] = 0
    used = set()
    while len(used) != len(G):
        min_d = float('+inf')
        for v in d:
            if d[v] < min_d and v not in used:
```

```
                current = v
                min_d = d[v]
        for neighbour in G[current]:
            l = d[current] + G[current][neighbour]
            if l < d[neighbour]:
                d[neighbour] = l
        used.add(current)
    return d # Алгоритм не эффективен
```

7. Реализация алгоритма Флойда-Уоршела

```
A = [[[INF]*n for i in range(n)] for k in range(n+1)] # INF - условная бесконечность, n - число ребер
for i in range(n):
    A[0][i][:] = W[i] # При копировании весовой матрицы W расстояние от вершины до себя равно нулю; забиваем матрицу рёбер т.е. расстояния в начальный момент.
for k in range(1, n+1):
    for i in range(n):
        for j in range(n):
            A[k][i][j] = min(A[k-1][i][j], A[k-1][i][k]+A[k-1][k][j]) # Добавляем путь от i до j вершины через новую вершину, если такой путь короче
```

8. Реализация алгоритма Прима

```
INF = 10**9 # Введем условную бесконечность
dist = [INF]*N # W[i][j] - вес ребра ij, который равен +бесконечность, если i не смежна j
dist[0] = 0
used = [False]*N
used[0] = True
tree = []
tree_weight = 0
for i in range(N):
    min_d = INF
    for j in range(N):
        if not used[j] and dist[j] < min_d:
            min_d = dist[j]
            u = j
    tree.append((i, u))
    tree_weight += min_d
    used[u] = True
    for v in range(N):
        dist[v] = min(dist[v], W[u][v])
```

9. Реализация алгоритма Краскала

```
N, M = [int(x) for x in input().split()]
edges = []
for i in range(M):
    v1, v2, weight = map(int, input().split())
```

```python
        edges.append((weight, v1, v2)) # Сначала будем добавлять вес
    edges.sort()
    comp = list(range(N))
    tree = []
    tree_weight = 0
    for weight, v1, v2 in edges:
        if comp[v1] != comp[v2]:
            tree.append((v1, v2))
            tree_weight += weight
            for i in range(N):
                if comp[i] == comp[v2]:
                    comp[i] = comp[v1]
```

10. **Топологическая сортировка** — упорядочивание вершин бесконтурного ориентированного графа согласно частичному порядку, заданному ребрами орграфа на множестве его вершин.

Топологическая сортировка, алгоритм Тарьяна

```python
Visited = [False]*(n + 1)
Ans = []

def DFS(start):
    Visited[start] = True
    for u in V[start]:
        if not Visited[u]:
            DFS(u)
    Ans.append(start)

for i in range(1, n + 1):
    if not Visited(i):
        DFS(i)
Ans = Ans[::-1]
```