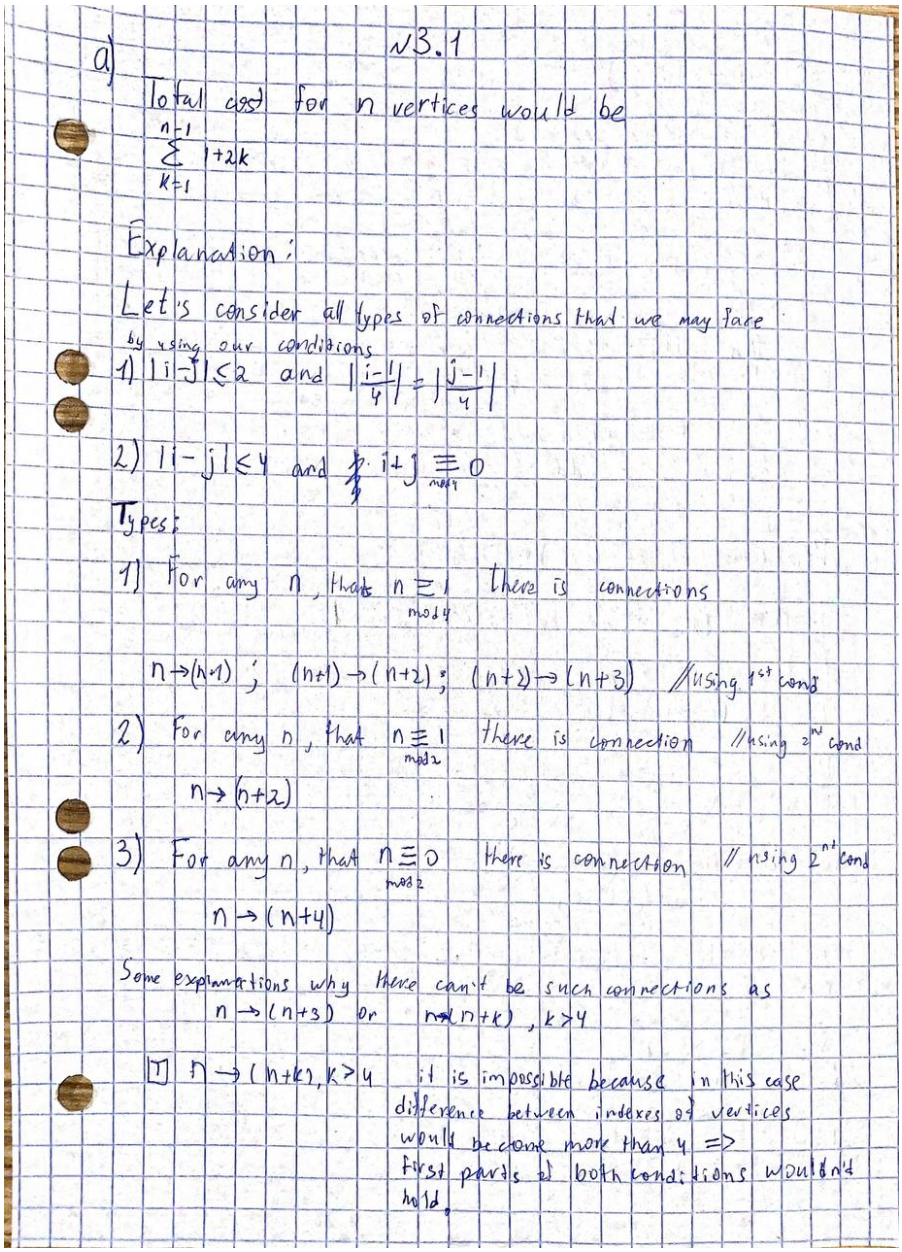
DSA Assignment 3 Theoretical Part Klyushev Vsevolod BS20-02

3.1 Minimum spanning tree



№3.1

a)

Total cost for n vertices would be

$\sum_{k=1}^{n-1} 1+2k$

Explanation:

Let's consider all types of connections that we may face by using our conditions

1) $|i-j| \leq 2$ and $\left|\frac{i-1}{4}\right| = \left|\frac{j-1}{4}\right|$

2) $|i-j| \leq 4$ and ~~$i+j \equiv 0 \pmod 4$~~

Types:

1) For any n, that $n \equiv 1 \pmod 4$ there is connections

$n \to (n+1)$; $(n+1) \to (n+2)$; $(n+2) \to (n+3)$ // using 1st cond

2) For any n, that $n \equiv 1 \pmod 2$ there is connection // using 2nd cond

$n \to (n+2)$

3) For any n, that $n \equiv 0 \pmod 2$ there is connection // using 2nd cond

$n \to (n+4)$

Some explanations why there can't be such connections as $n \to (n+3)$ or $n \to (n+k)$, $k > 4$

1) $n \to (n+k)$, $k > 4$ it is impossible because in this case difference between indexes of vertices would become more than 4 => first parts of both conditions wouldn't hold.

2. $n \to (n+3)$

It is impossible because:

1) first part of 1st condition wouldn't work, since difference between indexes is more than 2

2) Second part of 2nd condition wouldn't work, since:

Let i, j - be such vertices, that $j = i+3$,

then $i+j = 2i+3 \Rightarrow$ there is no way that $i+j = 2i+3 \equiv 0 \pmod 4$ because $2i$ would be always even and therefore $2i+3$ would be always odd.

Now let's look at weight function

~~let a and b be some vertices and d =~~

Lets consider some vertices $a, b \mid a<b$

Let $d = (b-a)$ - be difference, $d > 0$

Then applying formula of weight $w(i,j) = i+j + (|i-j|-1)^2$ we will have

$w(a,b) = a+b+(b-a-1)^2 =$

$= a+a+d+(d-1)^2 = 2a+d^2-d+1$

In case of our task d may be equal to 1, 2 or 4

Therefore:

- $d=1$ correlates with 1st type connection
  $w(a,b) = 2a+1$
- $d=2$ correlates with 2nd type connection
  $w(a,b) = 2a+3$
- $d=4$ correlates with 3rd type connection
  $w(a,b) = 2a+13$

So, if we were in building MST for graph G we would like to pick connections with $dist(a,b)=1$ but we can't do that every time because of 1 type connection limitations.

* So, let's divide the vertices of the graph into groups of 4, $(n; n+1; n+2; n+3 \mid n \underset{mod\ 4}{\equiv} 1)$ where there would be support of 1st type connection.

Now we need to connect that groups in most efficient way. So, if we can't use 1st type connections, let's look at 2nd one:

Lets consider 2 groups from * $(n \bmod 4 = 1)$

① $n; n+1; n+2; n+3$

② $n+4; n+5; n+6; n+7$

There are 3 ways to connect that groups

1 way: $(n+2) \rightarrow (n+4)$

it would use 2nd type of connection, and its weight would be

$2(n+2)+3 = 2n+7$

2 way: $(n+1) \rightarrow (n+5)$

it would use 3rd type of connection, and it's weight would be

$2(n+1)+13 = 2n+15$

3 way: $(n+3) \rightarrow (n+4)$

it would use 3rd type of connection, and it's weight would be

$2(n+3)+13 = 2n+19$

So, the most efficient way to connect that groups is connection of 2nd type

But lets return in way how we connect group

$(n; n+1; n+2; n+3 \mid n \underset{\text{mod}4}{\equiv} 1)$

There would be following possible connections:

$n \to (n+1)$ 1st type weight = $2n+1$
$n \to (n+2)$ 2nd type weight = $2n+3$; adding $(n+2)$ to $(n, n+1)$
$(n+1) \to (n+2)$ 1st type weight = $2n+3$;
$(n+1) \to (n+3)$ 2nd type weight = $2n+5$; adding $(n+3)$ to $(n, n+1, n+2)$
$(n+2) \to (n+3)$ 1st type weight = $2n+5$;

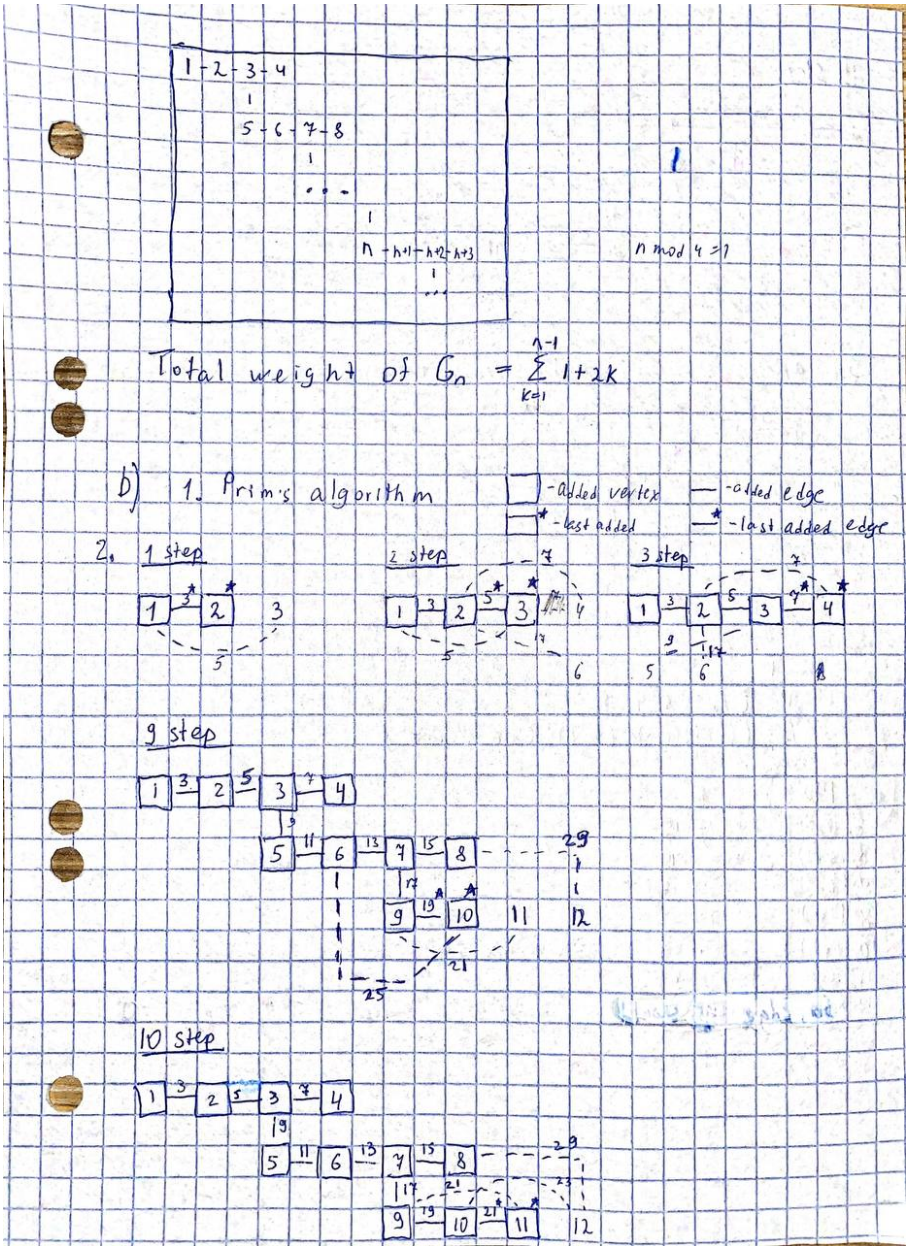
So there is no difference which type of connection we'll use in order to connect $(n; n+1; n+2; n+3)$, because total cost would be $(2n+1)+(2n+3)+(2n+5)$ anyway.

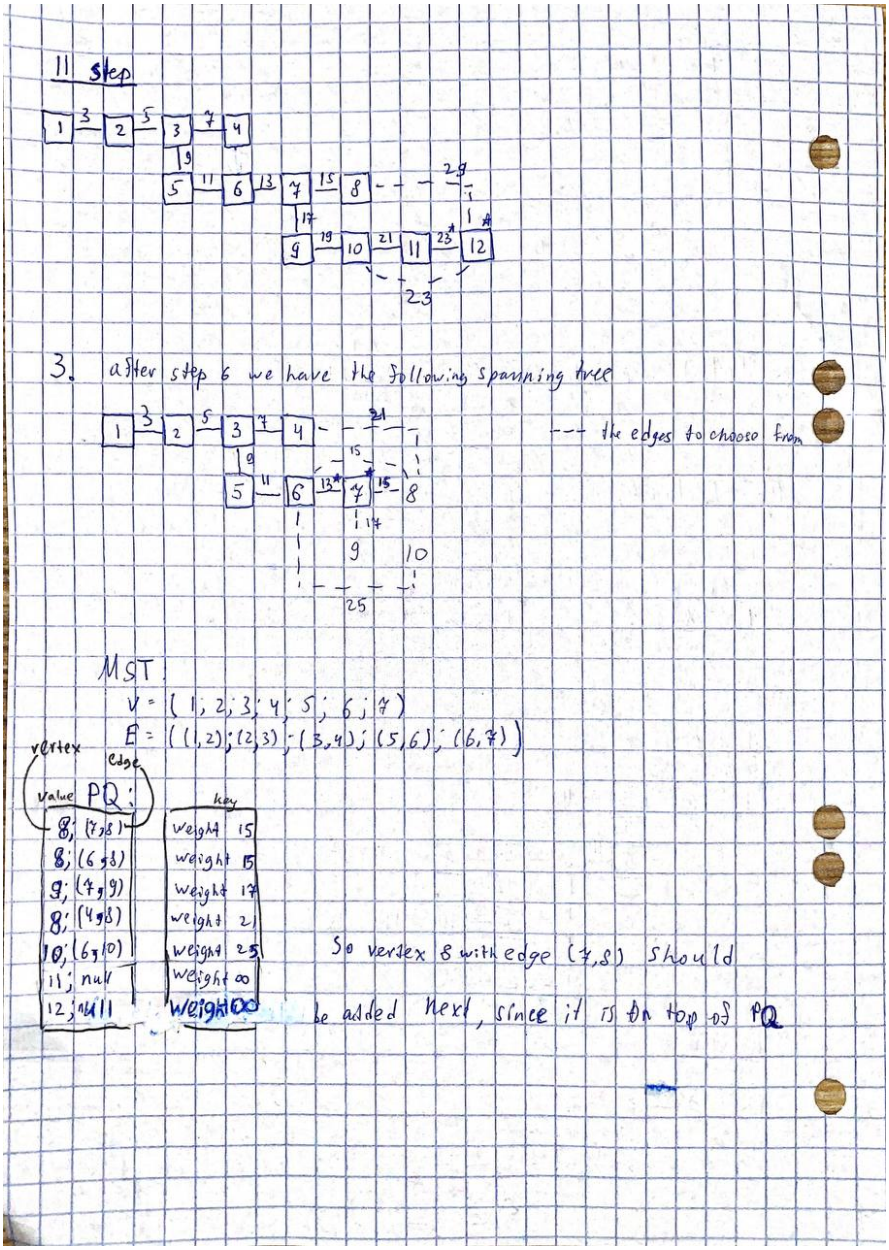
So, now let's show what happens if we want to connect $(n+4)$ with $(n; n+1; n+2; n+3)$

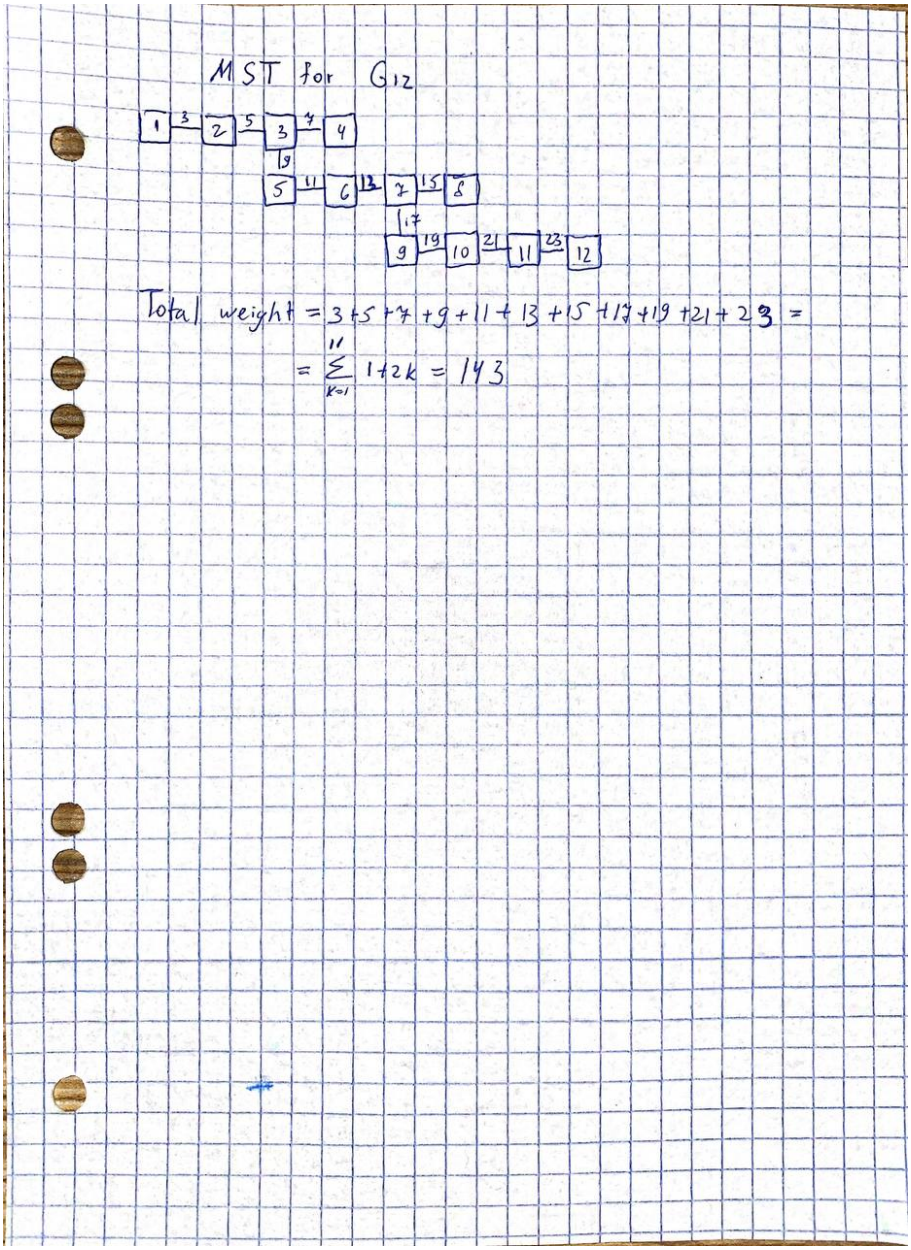
we already know that we need to use connection of 2nd type => total weight of $(n; n+1; n+2; n+3; n+4)$ would be:

$(2n+1)+(2n+3)+(2n+5)+(2n+7)$
Moreover $(n+4)$ may be beginning vertex of new group of 4.
So, it's easy to see that total weight would be just sum of odd numbers.
(each connection is odd numbe that is more than previous one by 2)

Therefore, we can came to conclusion, that for n vertices in G, total weight would be sum of n-1 odd numbers starting from 3


1-2-3-4
5-6-7-8
n-n+1-n+2-n+3
n mod 4 = 1
Total weight of $G_n = \sum_{k=1}^{n-1} 1+2k$
b) 1. Prim's algorithm
- added vertex
- last added
- added edge
- last added edge
2. 1 step
2 step
3 step
9 step
10 step


11 step
3. after step 6 we have the following spanning tree
--- the edges to choose from
MST:
V = {1; 2; 3; 4; 5; 6; 7}
E = {(1,2); (2,3); (3,4); (5,6); (6,7)}
vertex
edge
value
PQ:
key
8; (7,8)
8; (6,8)
9; (4,9)
8; (4,8)
10; (6,10)
11; null
12; null
weight 15
weight 15
weight 17
weight 21
weight 25
weight ∞
weight ∞
So vertex 8 with edge (7,8) should be added next, since it is on top of PQ



3.2 Shortest path

1.

In order to store the path we need to create one more matrix, where we store an index of vertex, from which we came from.

```
let dist be a 2-dimensional vector of minimum distances
let prev be a 2-dimensional vector of indexes of vertices, from which we came from
let V be a vector of vertices
let E be a vector of edges (u,v)

void initialization(n)
    dist[n][n]
    prev[n][n]
    for u from 1 to n
        for v from 1 to n
            dist[u][v] ← inf
            prev[u][v] ← 0
            if (u==v)
              dist[v][v] ← 0
              prev[v][v] ← 0
```

```
void floyd()
    for each edge (u, v) from E do
        dist[u][v] ← weight(u, v)
        prev[u][v] ← u
    for each vertex v from V do
        dist[v][v] ← 0
        prev[v][v] ← 0
    for k from 1 to V.length() do
        for i from 1 to V.length()
            for j from 1 to V.length()
                if dist[i][j] > dist[i][k] + dist[k][j] then
                    dist[i][j] ← dist[i][k] + dist[k][j]
                    prev[i][j] ← k

vector fPath(vector path, i, j)
    if (prev[i][j]=0)
        return path.reverse()
    path.pushback(j)
    return fPath(path,i,prev[i][j])
```

2.

if we want to add new vertex, we need to resize our matrices and copy all data from previous into them

```
let dist be a 2 dimensional vector of minimum distances
let prev be a 2 dimensional vector of indexes of vertices, from which we came from
let distC be a copy of dist
let prevC be a copy of prev

void copy(n)
    distC[n][n]
    prevC[n][n]
    for u from 1 to n
        for v from 1 to n
            distC[u][v] ← dist[u][v]
            prevC[u][v] ← prev[u][v]

void addVertex(k)
    copy(V.length())
    initialization(V.length()+1) //now dist is a (V+1)*(V+1) 2-dimensional vector of minimum distances initialized by infinity's, but
                      //now prev is a (V+1)*(V+1) 2-dimensional vector of indexes of vertices, from which we came from initialized b
    for u from 1 to V.length()
        for v from 1 to V.length()
            dist[u][v] ← distC[u][v]
            prev[u][v] ← prevC[u][v]
    V.pushback(k)

void addEdge(s,e)
    E.pushback((s,e))
    if (weight(s,e)<dist[s][e])
        dist[s][e] ← w(s,e)
        k ← s
        for i from 1 to V.length()
            for j from 1 to V.length()
                if dist[i][j] > dist[i][k] + dist[k][j] then
                    dist[i][j] ← dist[i][k] + dist[k][j]
                    prev[i][j] ← k
        k ← e
        for i from 1 to V.length()
            for j from 1 to V.length()
                if dist[i][j] > dist[i][k] + dist[k][j] then
                    dist[i][j] ← dist[i][k] + dist[k][j]
                    prev[i][j] ← k
```

3.

for n = number of vertices.

addVertex() has time complexity $T(n) = O(n^2) + O(n^2) + O(n^2) + const = O(n^2)$, because:

- copy() has $T(n) = O(n^2)$ (2 loops)
- initialization() has $T(n) = O(n^2)$ (2 loops)
- copying information from distC and prevC takes $T(n) = O(n^2)$ (2 loops)
- adding new vertex to a vector has $T(n) = const$

addEdge() has time complexity $T(n) = const + O(n^2) = O(n^2)$, because:

- adding new edge to a vector has $T(n) = const$
- updating all paths that contains new edge takes at worst case $T(n) = O(n^2) + O(n^2) = O(n^2)$ because of 2 times of 2 loops