Napad na NASP

Algoritmi obilaska stabla

BFS (Breadth First Search) – **O(V + E)**

-nerekurzivan, (struktura – lista)

```
BreadthFirstSearch (G)
inicijalizacija: svim vrhovima num(v)=0
korak = 0; ustrojiti pomoćni red spremnik;
while u grafu G još ima neobiđenih vrhova v //num(v)==0
     num(v) = ++korak;
     dodati v u red spremnik (na kraj);
     while još ima vrhova u redu spremnik //nije prazan
          prvi = prvi u redu;
          for svi neobiđeni susjedi u vrha prvi
               num(u) = ++korak;
               vrh u dodati u red spremnik;
               zabilježiti brid (prvi,u); //ako treba
                                          //pamtiti put
```

DFS (Depth First Search) – **O(V + E)**

-rekurzivan, (struktura – stog)

```
DepththFirstSearch (G)
inicijalizacija: svim vrhovima num(v)=0
korak = 0;          //je li potrebno ovisi o DFS(v)
while u grafu G još ima neobiđenih vrhova v //num(v)==0
     DFS(v);
return edges;

DFS (v)
num(v) = ++korak;   //korak:statička,argument,globalna
for svi susjedi u vrha v    //Uvjeti u 'for' i 'if' zapravo
     djeluju zajedno i izdvajaju samo neobiđene susjede
     if num(u)==0
          prethodnik(u) = v;       //ako je potrebno
          zabilježiti brid (v,u) u edges;
          DFS(u);
```

Shortest path algoritmi

Dijkstra – $O(V^2)$ ili sa heapom - – $O(E + V*\log_2 V)$

```
Dijkstra (source, dest)
initialization: for all vertices d(v)= inf   // d(v) = d(source,v)
d(source) = 0;
ToBeCh = all vertices;     //ToBeCh - popis svih neobrađenih
while there are vertices in ToBeCh     //... is not empty
      v = a vertex in ToBeCh with the least d(v);
      if v == dest     //Bez ovog uvjeta naći će najmanje
            return ... ;     //udaljenosti do svih.
      remove v from ToBeCh;
      for all vertices u adjacent to v and in ToBeCh
            dnew(u)=d(v)+edge(v,u); //moguća nova d(u)
            if dnew(u) < d(u)
                  d(u) = dnew(u);
                  predecessor(u) = v;      //prethodnik se
                  //najlakše čuva kao članska varijabla vrha
```

Bellman-Ford – $O(V*E)$

```
BellmanFord (graph)
initialisation: for all vertices d(v)= inf   // d(v) = (source,v)
d(source) = 0;
while there is an edge(u,v) such that
                        d(u)+edge(u,v) < current d(v)
      d(v) = d(u)+edge(u,v);
      predecessor(v) = u;
```

WFI (Warshall-Floyd-Ingerman) – $O(V^3)$

```
WFI (W)          //W=adjacency matrix of the graph
initialization of matrix D;      //matrica udaljenosti D⁰ = W
initialization of matrix Path;     //matrica puteva π_ij⁰
for k = 1 to |V|                  //skup međuvrhova
      for i = 1 to |V|            //polazni vrh
            for j = 1 to |V|          //završni vrh
                  if D[i,k] + D[k,j] < D[i,j]
                        D[i,j] = D[i,k] + D[k,j];
                        Path(i,j) = Path(k,j);
```

MST – Minimum Spanning Tree algoritmi

Kruskal – $O(E*\log_2 V)$

```
Kruskal (graph)
tree = null;                    //inicijalizacija MST
edges = all edges of graph sorted by weight;     //uzlazno
for (i=1; i≤|E| and |tree|<|V|-1; ++i)
    if e_i from edges does not form a cycle with edges in tree
            add e_i to tree;
```

Prim– $O(E + V*\log_2 V)$

```
Prim (graph)
newG = arbitrary starting vertex;        //pomoćni skup vrhova
tree = null;                             //skup bridova MST
while (number of vertices in newG) < |V|
      choose a vertex v from graph (an edge(u,v)),
            which is closest to vertices in newG;
      add v to newG;
      add edge(u,v) to tree;      //'u' is the vertex
          //in newG to which v is the closest neighbour
```

Dijkstra MST – $O(E*V)$

```
DijkstraMST (graph)
tree = null;                  //inicijalizacija MST
edges = all edges of graph;   //bilo koji poredak
for i=1 to |E|
      add e_i to tree;
      if there is a cycle in tree
            remove the maximum weight edge from the cycle;
```