Universiteti i Prishtinës

Fakulteti i Inxhinierisë Elektrike dhe Kompjuterike

Algoritmet dhe struktura e të dhënave

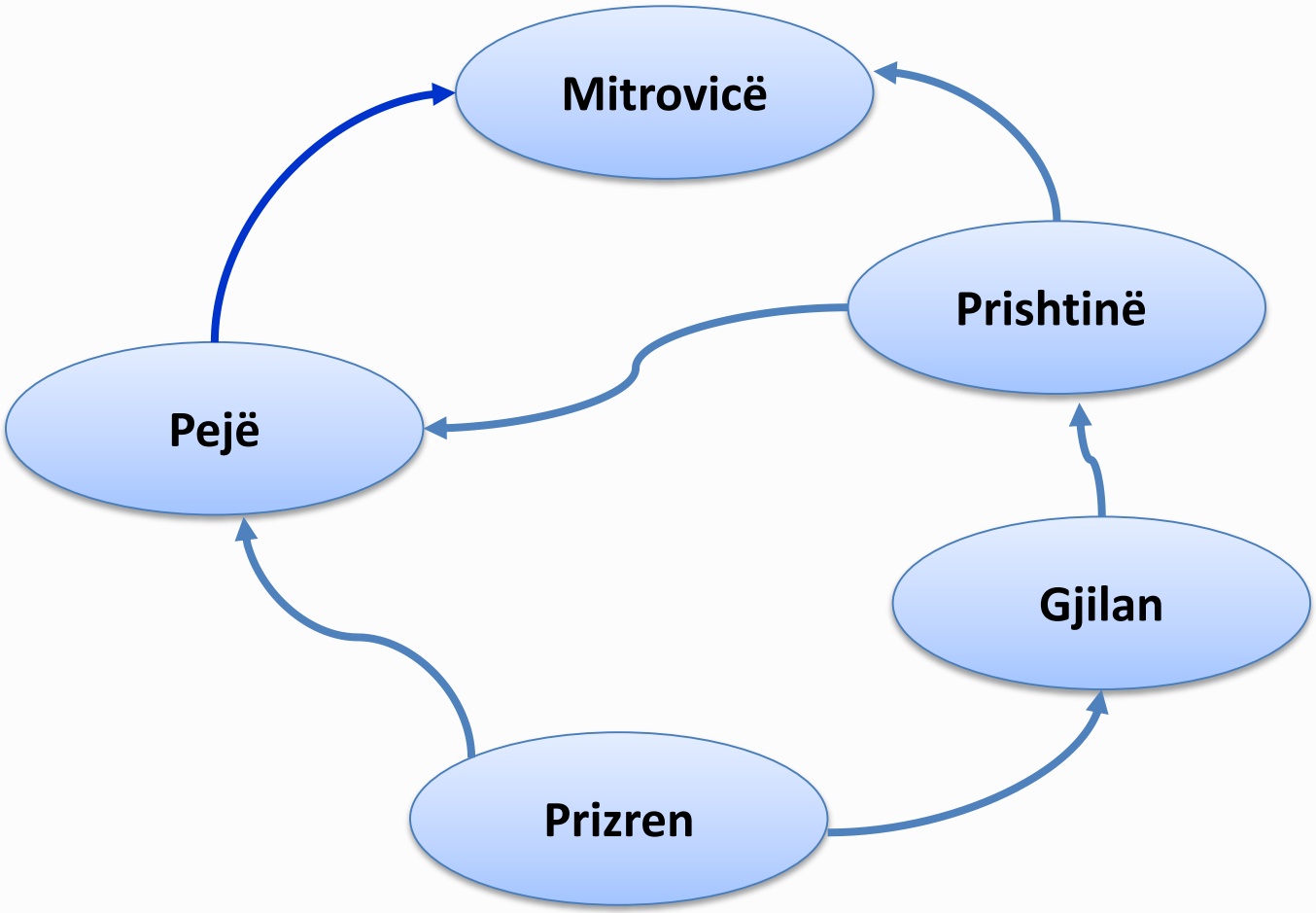
Vehbi Neziri

FIEK, Prishtinë 2015/2016

- Grafet
  - Algoritmet themelore të grafeve/ fqinjët
  - Përshkimi i grafit
  - Pema e shtrirjes minimale
  - Shtegu më i shkurtër – Alg. i Dijkstras
  - Matrica e fqinjësisë
  - Lista e fqinjësisë

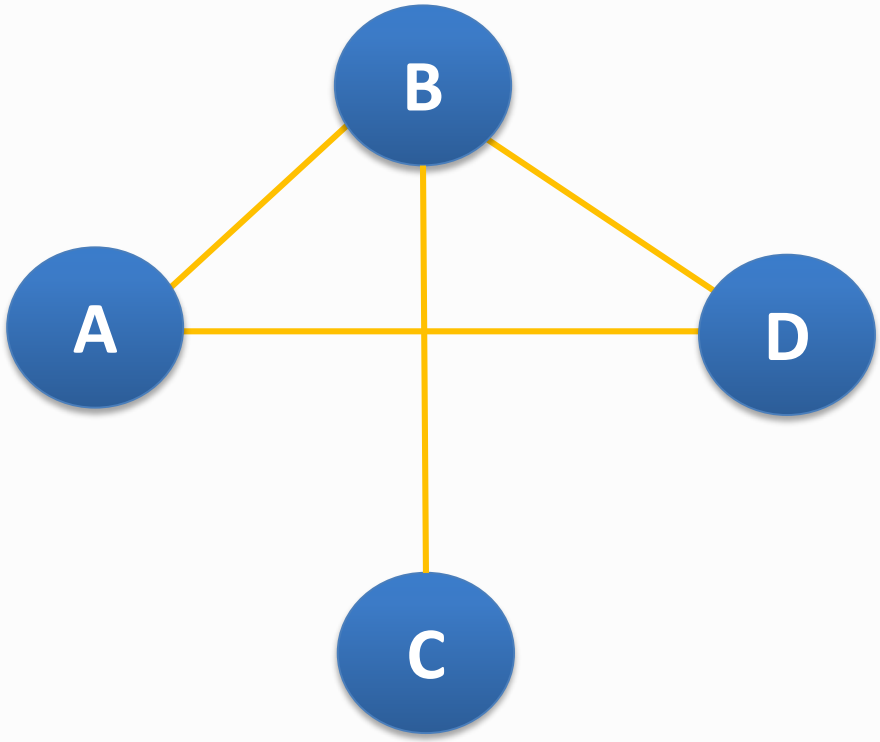
Grafet

- Grafi është koleksion i nyjave dhe i degëve që lidhin nyjat me njëra tjetrën.



Grafet

- Grafi **G** është koleksion **V(G)** i nyjave dhe i degëve **E(G)** që lidhin nyjat me njëra tjetrën, pra **G=(V,E)**



V = {A, B, C, D}

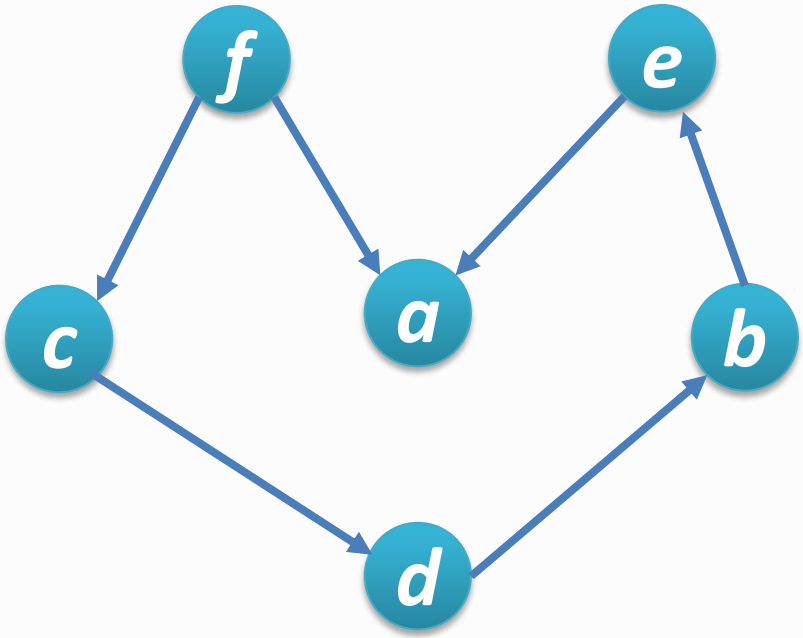
***E* = { (A, B), (A, *D*), (B, *C*), (B, D) }**

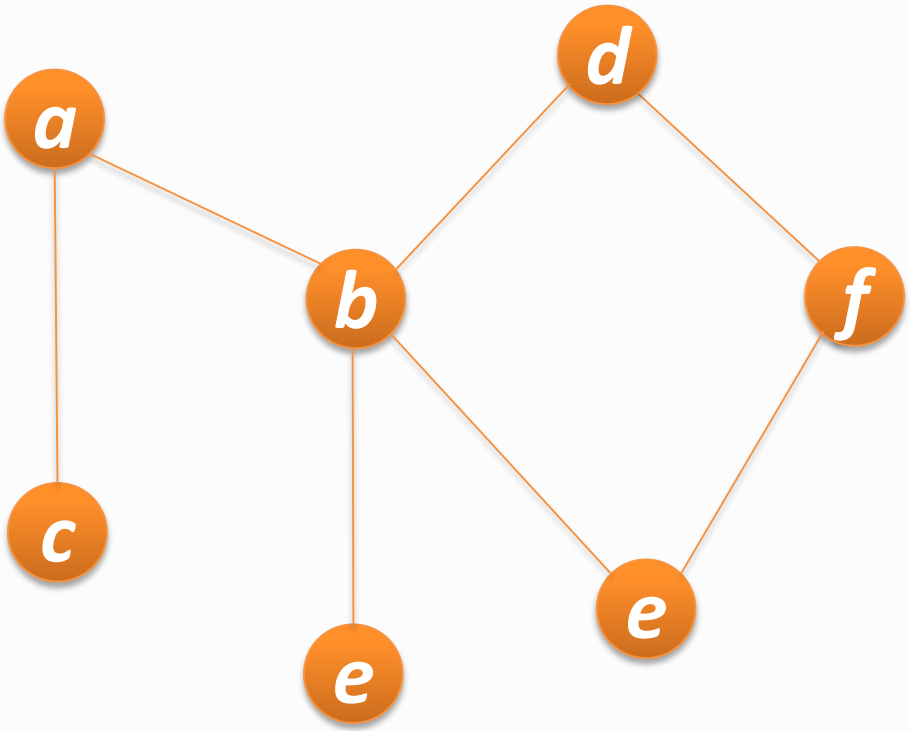
Përdorimi i grafeve

- Internet
- Trafik rrugor
- Rrjete sociale
- Gjeometri
- Në shumë sfera tjera…

- Grafi mund të jetë
  - i drejtuar (degët/rrugët mund të përshkohen vetëm në njërin drejtim - digraf)
  - i padrejtuar (degët/rrugët mund të përshkohen në dy drejtimet)

Grafi i drejtuar dhe i padrejtuar





Shembull 12.1

- Në këtë shembull do të tregohet implementimi i grafit dhe funksionet themelore të grafeve.

```cpp
#include <iostream>
using namespace std;

struct NyjaFqinje
{
    int destinacioni;
    NyjaFqinje *vijuese;
};

struct Fqinjet
{
    NyjaFqinje *fillimi;
};

struct Grafi
{
    int numri;
    Fqinjet *vargu;
};

NyjaFqinje *NyjeRe(int dest)
{
    NyjaFqinje *nyRe = new (NyjaFqinje);
    nyRe->destinacioni = dest;
    nyRe->vijuese = NULL;
    return nyRe;
}

Grafi *krijoGrafin(int nr) //krijo grafin me 'nr' nyje
{
    Grafi *grafi = new Grafi;
    grafi->numri = nr;

    grafi->vargu = new Fqinjet[nr * sizeof(Fqinjet)];

    for (int i = 0; i < nr; ++i) //inico listen si te zbrazur
        grafi->vargu[i].fillimi = NULL;

    return grafi;
}

void shtoDegen(Grafi *grafi, int bur, int dest)//graf i padrejtuar
{
    //shto degen nga burimi ne destinacion
    NyjaFqinje *nyRe = NyjeRe(dest);
    nyRe->vijuese = grafi->vargu[bur].fillimi;
    grafi->vargu[bur].fillimi = nyRe;

    //shto edhe rrugen e anasjelltë nga destinacioni ne burim
    nyRe = NyjeRe(bur);
    nyRe->vijuese = grafi->vargu[dest].fillimi;
    grafi->vargu[dest].fillimi = nyRe;
}

void shtypGrafin(Grafi *grafi)
{
    int nyje;
    for (nyje = 0; nyje < grafi->numri; ++nyje)
    {
        NyjaFqinje *dega = grafi->vargu[nyje].fillimi;
        cout << "\nFqinjet e nyjes " << nyje << ": ";
        while (dega)
        {
            cout << " " << dega->destinacioni << " - ";
            dega = dega->vijuese;
        }
        cout << "\n";
    }
}

int main()
{
    int skaje = 6;
    struct Grafi *grafi = krijoGrafin(skaje);
    shtoDegen(grafi, 0, 1);
    shtoDegen(grafi, 0, 3);
    shtoDegen(grafi, 1, 2);
    shtoDegen(grafi, 1, 5);
    shtoDegen(grafi, 2, 4);
    shtoDegen(grafi, 2, 3);
    shtoDegen(grafi, 3, 4);
    shtoDegen(grafi, 5, 4);
    shtoDegen(grafi, 5, 0);

    shtypGrafin(grafi);
    system("Pause");
    return 0;
}
```

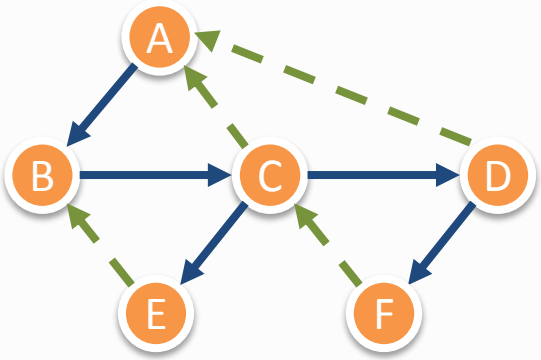
Shembull 12.2

- Në këtë shembull do të tregohet implementimi i grafeve përmes distancës në mes qyteteve.

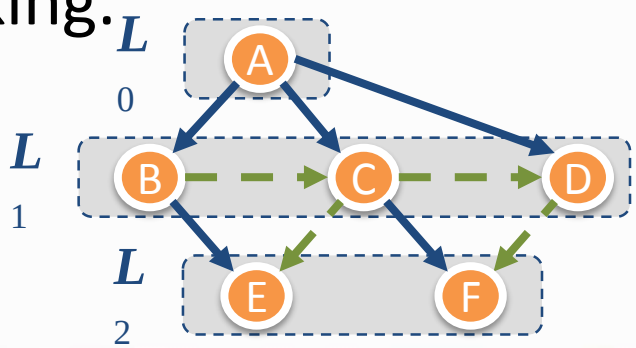
```
1  #include <vector>
2  #include <string>
3  #include <iostream>
4  using namespace std;
5
6  class Nyjat;
7
8  class Deget
9  {
10 private:
11     Nyjat* origjina;
12     Nyjat* destinacioni;
13     int distanca;
14
15 public:
16     Deget(Nyjat *org, Nyjat *dest, int dist)
17     {
18         origjina = org;
19         destinacioni = dest;
20         distanca = dist;
21     }
22
23     Nyjat* lexo_origjina()
24     {
25         return origjina;
26     }
27     Nyjat* lexo_destinacioni()
28     {
29         return destinacioni;
30     }
31     int lexo_distanca()
32     {
33         return distanca;
34     }
35 };
```

Përshkrimi i grafit

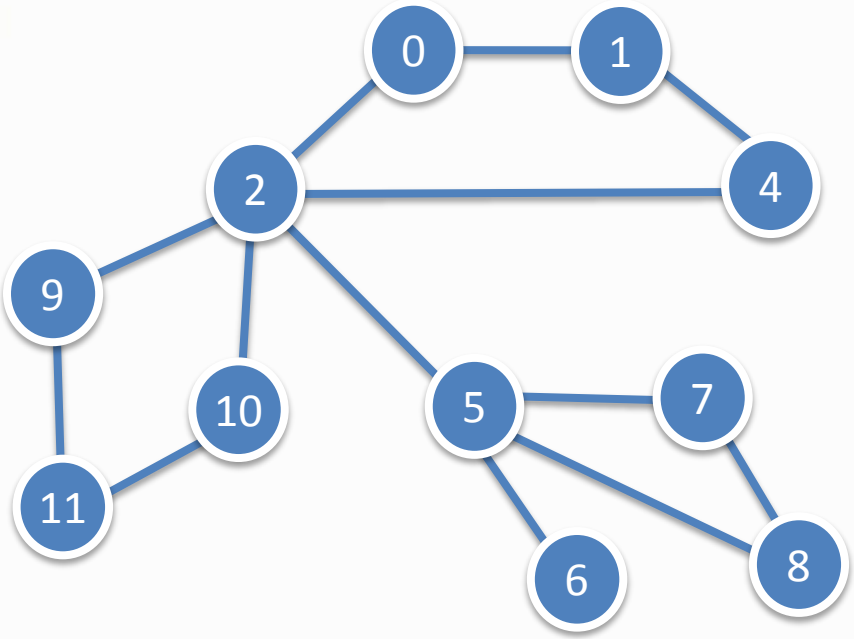
- Breadth-First Search (BFS)
  - Algoritëm për përshkim dhe kërkim të struktures së të dhënave në pemë ose graf.



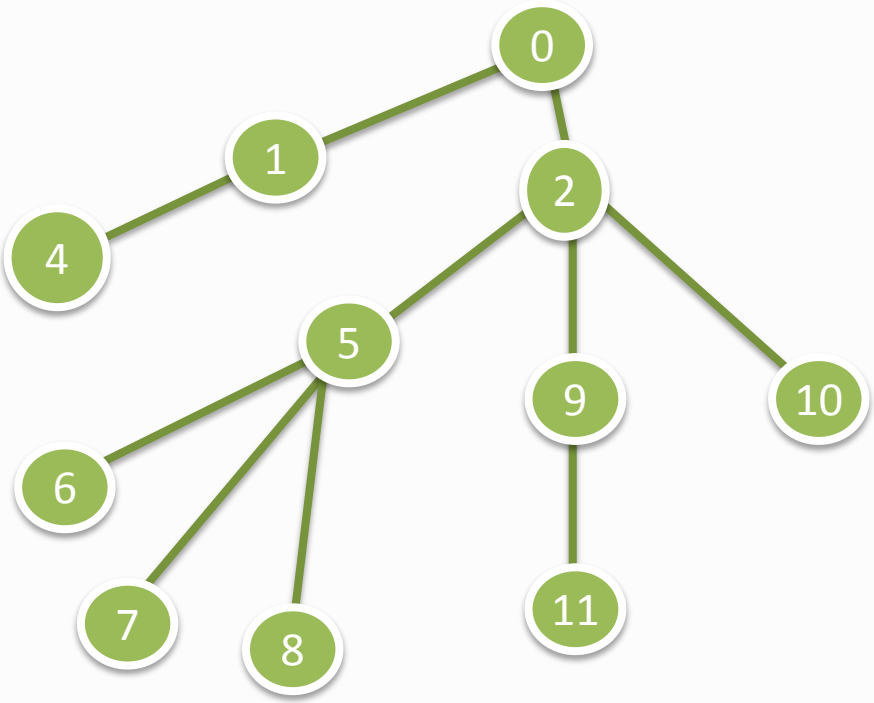
- Depth-First Search (DFS)
  - Algoritëm për përshkim dhe kërkim në pemë ose graf. Fillon në rrënjë dhe eksploron aq sa është e mundur para se të bëjë backtracking.



BFS

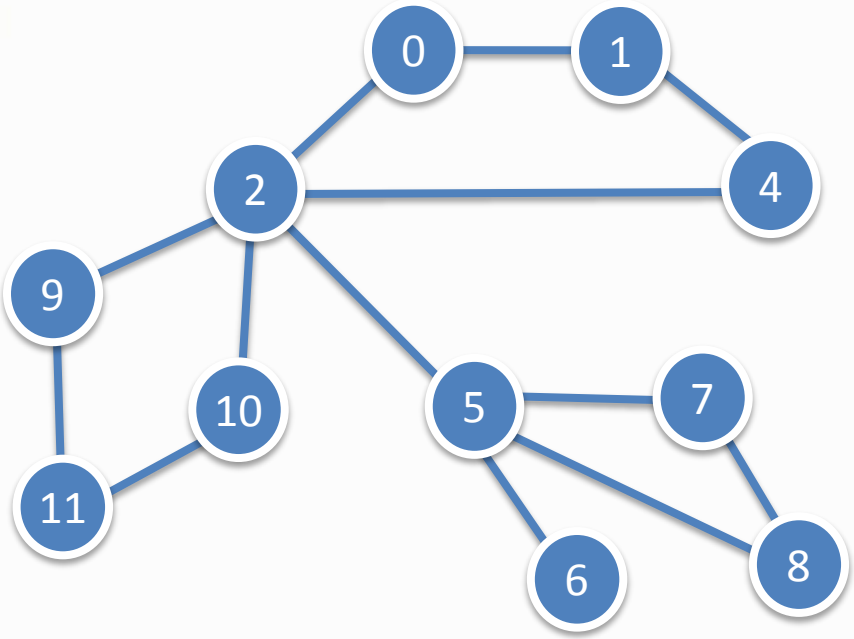


Grafi G

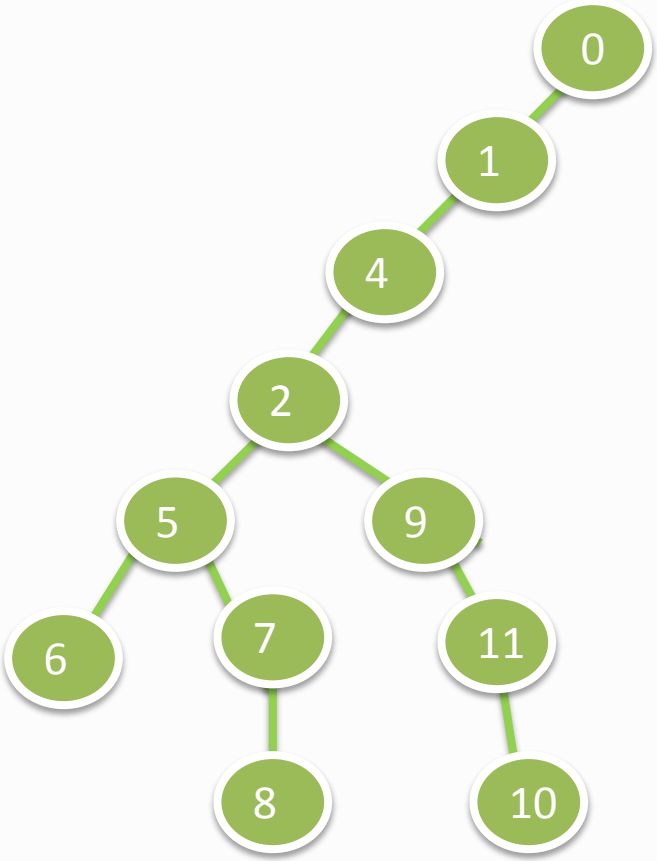


Pema BFS

DFS



Grafi G



Pema DFS

Shembull 12.3

- Në këtë shembull do të tregohet algoritmi BFS.

```
56 int main()
57 {
58     Grafi g(4);
59     g.shtoDegen(0, 1);
60     g.shtoDegen(0, 2);
61     g.shtoDegen(1, 2);
62     g.shtoDegen(2, 0);
63     g.shtoDegen(2, 3);
64     g.shtoDegen(3, 3);
65
66     cout << "Peshkimi BFS - (duke filluar nga nyja 2) \n\n";
67     g.BFS(2);
68     cout << endl << endl;
69
70     system("Pause");
71     return 0;
72 }
```

```
1  #include<iostream>
2  #include <list>
3  using namespace std;
4
5  class Grafi
6  {
7      int skaje;
8      list<int> *fqinje;
9  public:
10     Grafi(int V);
11     void shtoDegen(int v, int u);
12     void BFS(int s);  //BFS (Gjeresia se pari)
13 };
14
15 Grafi::Grafi(int nr)
16 {
17     this->skaje = nr;
18     fqinje = new list<int>[nr];
19 }
20
21 void Grafi::shtoDegen(int v, int u)
22 {
23     fqinje[v].push_back(u); // Shto u ne listen e v-së.
24 }
25
26 void Grafi::BFS(int s)// (Breadth First Search)
27 {
28     bool *eVizituar = new bool[skaje];
29     for (int i = 0; i < skaje; i++)
30         eVizituar[i] = false;
31
32     list<int> queue;
33
34     eVizituar[s] = true;
35     queue.push_back(s);
36
37     list<int>::iterator i;
38
39     while (!queue.empty())
40     {
41         s = queue.front(); //tërhiqe nga queue
42         cout << s << " ";
43         queue.pop_front();
44
45         for (i = fqinje[s].begin(); i != fqinje[s].end(); ++i)
46         {
47             if (!eVizituar[*i])
48             {
49                 eVizituar[*i] = true;
50                 queue.push_back(*i);
51             }
52         }
53     }
54 }
```

Shembull 12.4

- Në këtë shembull do të tregohet algoritmi DFS.

```
1  #include<iostream>
2  #include <list>
3  using namespace std;
4
5  class Grafi
6  {
7      int skaje;
8      list<int> *fqinje;
9      void DFSNdihmes(int v, bool eVizituar[]);
10 public:
11     Grafi(int V);
12     void shtoDegen(int v, int u);
13     void DFS();
14 };
15
16 Grafi::Grafi(int nr)
17 {
18     this->skaje = nr;
19     fqinje = new list<int>[nr];
20 }
21
22 void Grafi::shtoDegen(int v, int u)
23 {
24     fqinje[v].push_back(u);
25 }
26
27 void Grafi::DFSNdihmes(int v, bool eVizituar[])
28 {
29     eVizituar[v] = true;
30     cout << v << " ";
31
32     // Rikthehu per te gjitha nyjet fqinje te kesaj nyjeje
33     list<int>::iterator i;
34     for (i = fqinje[v].begin(); i != fqinje[v].end(); ++i)
35         if (!eVizituar[*i])
36             DFSNdihmes(*i, eVizituar);
37 }
38
39 void Grafi::DFS()
40 {
41     bool *eVizituar = new bool[skaje];
42     for (int i = 0; i < skaje; i++)
43         eVizituar[i] = false;
44
45     for (int i = 0; i < skaje; i++) //shtyp pershkrimin DFS
46         if (eVizituar[i] == false)
47             DFSNdihmes(i, eVizituar);
48 }
49
50 int main()
51 {
52     Grafi g(4);
53     g.shtoDegen(0, 1);
54     g.shtoDegen(0, 2);
55     g.shtoDegen(1, 2);
56     g.shtoDegen(2, 0);
57     g.shtoDegen(2, 3);
58     g.shtoDegen(3, 3);
59
60     cout << "Pershkimi DFT(duke filluar nga nyja 0) \n\n";
61     g.DFS();
62     cout << endl << endl;
63     system("Pause");
64     return 0;
65 }
```

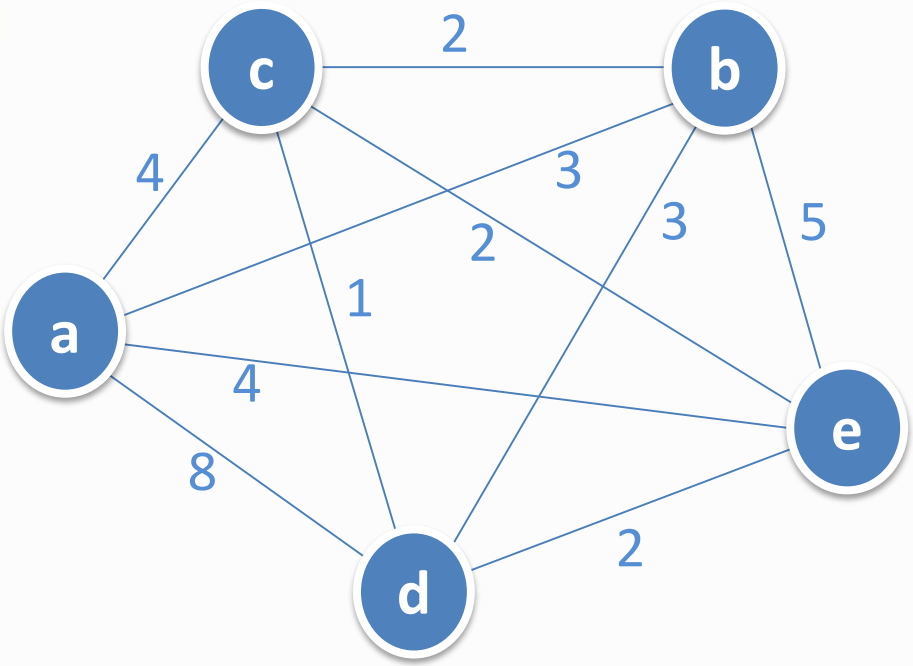
Pema e shtrirjes minimale

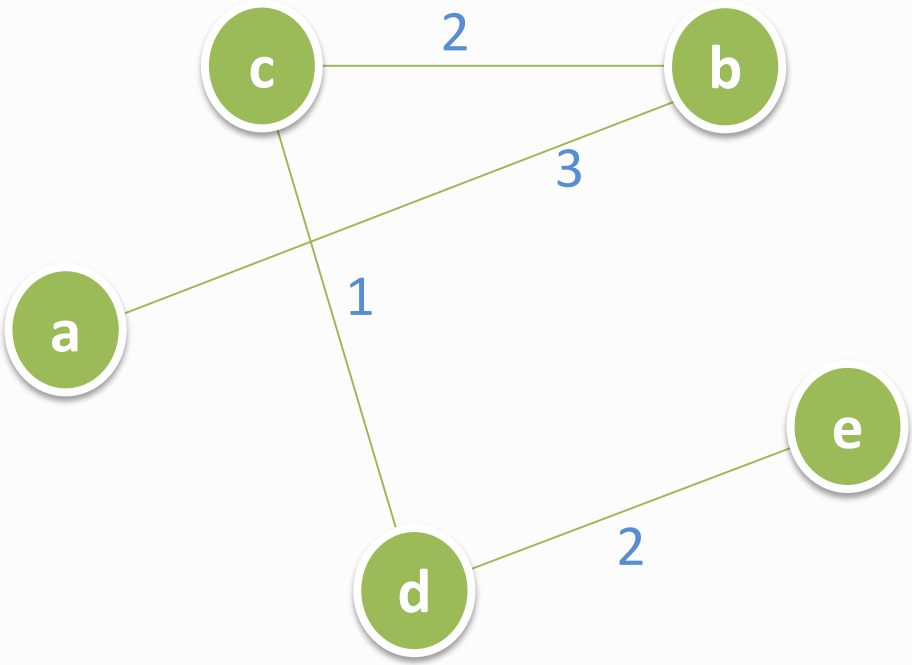
- Pema e shtrirjes së një grafi të pa drejtuar G është nën-grafi i **G**-së i cili përmban të gjitha nyjat e G.

- Në një graf të peshuar, pesha e një nën grafi është shuma e peshave të degëve në nën graf.

- Pema e shtrirjes minimale (MST) për një graf të peshuar të pa drejtuar është pemë e shtrirjes ku shuma e peshave për degët ka peshën minimale.

Pema e shtrirjes minimale





Grafi i pa drejtuar dhe pema e shtrirjes minimale.

Algoritmi Dijkstra-Prim

- Algoritmi për gjetjen e pemës së shtrirjes minimale u zhvilluar nga Dijkstra dhe Prim në fund të viteve ‘50.
- Ata përdoren një algoritëm lakmitar/pa ngopur të quajtur “greedy”

```
Përzgjidh nyjën fillestare
Ndërto “vijat” nga nyja fillestare
While (Gjersa) ka akoma nyje të mbetura, përzgjidh
degën me peshën me të vogël
    • shto nyjën e shoqëruar/bashkuar/lidhur në pemë
    • Përditëso “vijat” duke:
        • shto nyjat në periferi (të lidhura) me nyjën
          e re
        • Përditëso degët ashtu që të jenë më të voglat
end while
```

Shembull 12.5

- Në këtë shembull do të tregohet algoritmi “greedy” për shtrirjen minimale (Prim’s Minimum Spanning Tree)

```
#include<iostream>
using namespace std;

#define N 5     // Numri i nyjave

int VleraMin(int vlera[], bool grupiMST[])
{
    int min = INT_MAX, min_index;

    for (int nyja = 0; nyja < N; nyja++)
        if (grupiMST[nyja] == false && vlera[nyja] < min)
            min = vlera[nyja], min_index = nyja;
    return min_index;
}

void printoMST(int prindi[], int n, int grafi[N][N])
{
    cout << "Dega    Pesha\n";
    for (int i = 1; i < N; i++)
        cout << prindi[i] << " - " << i << "    " << grafi[i][prindi[i]] << "\n";

}

void primMST(int graph[N][N])
{
    int konstruktuar[N]; // per te ruajtur vlerat e konstruktuara MST
    int vlera[N];       // vlerat e perdorura me peshe minimale
    bool grupiMST[N];  // prezantimi i nyjave qe nuk jane perfshire ende ne MST

    for (int i = 0; i < N; i++) // Inico te gjitha vlerat ne MAx INT
        vlera[i] = INT_MAX, grupiMST[i] = false;

    // Perfshi gjithmonë nyjen e parëe në MST
    vlera[0] = 0;     // behet zero dhe merret si nyja e parë.
    konstruktuar[0] = -1; // Nyja e parë është rrënja e MST

    // MST do të ketë N nyja
    for (int count = 0; count < N - 1; count++)
    {
```

Shembull 12.5

- Në këtë shembull do të tregohet algoritmi “greedy” për shtrirjen minimale (Prim’s Minimum Spanning Tree)

```
        // Përzgjidh nyjen minimale prej bashkesisë së nyjeve që nuk janë përfshirë ende
        int u = VleraMin(vlera, grupiMST);
        // Nyjen e perzgjedhur shto ne grupin/bashkësinë MST
        grupiMST[u] = true;
        // Përditëso vleren dhe indeksin e prindit të nyjave fqinje për nyjen e përzgjedhur
        // Shqyrto vetem nyjat që ende nu janë përfshirë në MST
        for (int v = 0; v < N; v++)
            // grafi[u][v] është jo-zero vetëm për nyjat fqinje te m
            // grupiMST[v] eshte false për nyjat që ende nuk janë perfshirë në MST
            // përditëso vleren vetem nëse grafi[u][v] është me e vogël sesa vlera[v]
            if (graph[u][v] && grupiMST[v] == false && graph[u][v] <  vlera[v])
                konstruktuar[v] = u, vlera[v] = graph[u][v];
    }
    printoMST(konstruktuar, N, graph);
}

int main()
{
    /* Të krijohet grafi i mëposhtëm
    2    3
    (0)--(1)--(2)
    |   / \   |
    6| 8/   \5 |7
    | /     \ |
    (3)-------(4)
    9          */
    int grafi[N][N] = {
        { 0, 2, 0, 6, 0 },
        { 2, 0, 3, 8, 5 },
        { 0, 3, 0, 0, 7 },
        { 6, 8, 0, 0, 9 },
        { 0, 5, 7, 9, 0 }};

    primMST(grafi);

    system("pause");
    return 0;
}
```

Shembull 12.6

- Në këtë shembull do të tregohet algoritmi “greedy” për shtrirjen minimale (Kruskal’s Minimum Spanning Tree)

```
// Algoritmi Kruskal
#include<iostream>
using namespace std;
#define NY 5        // Numri i nyjave

struct Nyja // structura për prezantim të degëve të peshuara në graf
{
    int burimi, dest, pesha;
};

struct Grafi // për prezantim të grafit të lidhur, të padrejtuar dhe të peshuar
{
    int N, D;       // N-> nr i nyjave, D-> nr i degeve
    Nyja* ny;
};

Grafi* krijoGraf(int N, int D)  // Krijo grafin me N nyja dhe D degë
{
    Grafi* gr = (Grafi*)malloc(sizeof(Grafi));
    gr->N = N;
    gr->D = D;
    gr->ny = (Nyja*)malloc(gr->D * sizeof(Nyja));

    return gr;
}

struct nengrupi // për prezantim të nën grupit për union-find
{
    int prindi;
    int rangu;
};

int Find(nengrupi ng[], int i)//për gjetjen e grupit të një elementi i
{
    if (ng[i].prindi != i)      // gjej rrenjen dhe beje si prind te i
        ng[i].prindi = Find(ng, ng[i].prindi);

    return ng[i].prindi;
}
```

Shembull 12.6

- Në këtë shembull do të tregohet algoritmi “greedy” për shtrirjen minimale (Kruskal’s Minimum Spanning Tree)

```
void Union(nengrupi ng[], int x, int y) // union te dy grupeve te x dhe y, union sipas rangut
{
    int xroot = Find(ng, x);
    int yroot = Find(ng, y);

    // bashkangjit pemen me rangun me te vogel nën pemën me rangun më të lartë
    if (ng[xroot].rangu < ng[yroot].rangu)
        ng[xroot].prindi = yroot;
    else if (ng[xroot].rangu > ng[yroot].rangu)
        ng[yroot].prindi = xroot;
    else
    {
        ng[yroot].rangu = xroot;
        ng[xroot].rangu++;
    }
}

int Krahaso(const void* a, const void* b)
{
    struct Nyja* a1 = ( Nyja*)a;
    struct Nyja* b1 = ( Nyja*)b;
    return a1->pesha > b1->pesha;
}

void KruskalMST(Grafi* gr)
{
    int N = gr->N;
    Nyja result[NY];
    int e = 0, i = 0;

    qsort(gr->ny, gr->D, sizeof(gr->ny[0]), Krahaso);

    nengrupi *neng = (nengrupi*)malloc(N * sizeof(neng));

    for (int v = 0; v < N; ++v)
    {
        neng[v].prindi = v;
        neng[v].rangu = 0;
    }

    while (e < N - 1)
    {
        struct Nyja next_edge = gr->ny[i++];

        int x = Find(neng, next_edge.burimi);
        int y = Find(neng, next_edge.dest);

        if (x != y)
        {
            result[e++] = next_edge;
            Union(neng, x, y);
        }
    }

    cout<<"Ne vijim jane deget ne MST te konstruktuar\n";
    for (i = 0; i < e; ++i)
        cout << result[i].burimi << " - " << result[i].dest << " = " << result[i].pesha <<endl;
}
```

Shembull 12.6

- Në këtë shembull do të tregohet algoritmi “greedy” për shtrirjen minimale (Kruskal’s Minimum Spanning Tree)

```
100 int main()
101 {
102     /* Krijo pemen
103     10
104     0--------1
105     |  \     |
106     6|   5\   |15
107     |      \ |
108     2--------3
109     4       */
110     int N = 4;
111     int D = 5;
112     Grafi* gra = krijoGraf(N, D);
113
114     gra->ny[0].burimi = 0;
115     gra->ny[0].dest = 1;
116     gra->ny[0].pesha = 10;
117
118     gra->ny[1].burimi = 0;
119     gra->ny[1].dest = 2;
120     gra->ny[1].pesha = 6;
121
122     gra->ny[2].burimi = 0;
123     gra->ny[2].dest = 3;
124     gra->ny[2].pesha = 5;
125
126     gra->ny[3].burimi = 1;
127     gra->ny[3].dest = 3;
128     gra->ny[3].pesha = 15;
129
130     gra->ny[4].burimi = 2;
131     gra->ny[4].dest = 3;
132     gra->ny[4].pesha = 4;
133
134     KruskalMST(gra);
135
136     system("pause");
137     return 0;
138 }
```

Algoritmi i Dijkstra-s

- Single-Source Shortest Path Problem – problemi i gjetjes së rrugës më të shkurtër nga nyja burim tek të gjitha nyjat tjera në graf.

- Përdorimi
  - Koha e fluturimit të aeroplanëve
  - Kostoja e komunikimit telefonik
  - Koha e përgjigjes në rrjeta kompjuterike
  - Agjentët tregtar
  - …

Shembull 12.7

- Në këtë shembull do të tregohet algoritmi i Dijkstra-s.

```
#include <iostream>
using namespace std;

#define N 9

int DistancaMin(int dist[], bool bashkesiaSPT[])
{
    int min = INT_MAX, indeksi_min;

    for (int v = 0; v < N; v++)
        if (bashkesiaSPT[v] == false && dist[v] <= min)
            min = dist[v], indeksi_min = v;

    return indeksi_min;
}

void printSolution(int dist[], int n)
{
    printf("Nyja        Distanca nga burimi\n");
    for (int i = 0; i < N; i++)
        cout << i << "\t\t" << dist[i] << "\n";
}

void dijkstra(int grafi[N][N], int burimi)
{
    int dist[N];

    bool bashkesiaSPT[N];
    // ose distanca minimale prej burimit tek i eshte finalizuar

    // Inicializo te gjitha distancat si INFINITE dhe bashkesiaSPT[] si "false"
    for (int i = 0; i < N; i++)
        dist[i] = INT_MAX, bashkesiaSPT[i] = false;

    // Distanca e nyjes burimore perj vet-vetes eshte gjithmone 0
    dist[burimi] = 0;

    // Gjeje shtegun me te shkurter per te gjitha nyjet
    for (int nr = 0; nr < N - 1; nr++)
    {
        // Merre nyjen e distances minimale nga bashkesia e nyjeve te pa-procesuara
        // u eshte gjithmone baraz me "burimi" ne iteracionin e pare.
        int u = DistancaMin(dist, bashkesiaSPT);

        // Marko nyjen e zgjedhur si te procesuar
        bashkesiaSPT[u] = true;

        // Azhuro vleren e distances se nyjeve fqinje te nyjes se zgjedhur
        for (int v = 0; v < N; v++)

        // Azhuro dist[v] vetem nese nuk eshte ne "bashkesiaSPT", ekziston nje dege prej
        // u tek v, dhe pesha totale e shtegut prej burimi tek  v neper u eshte
        // me e vogel sesa vlera aktuale e dist[v]
            if (!bashkesiaSPT[v] && grafi[u][v] && dist[u] != INT_MAX
                && dist[u] + grafi[u][v] < dist[v])
                dist[v] = dist[u] + grafi[u][v];
    }

    printSolution(dist, N);
}
```

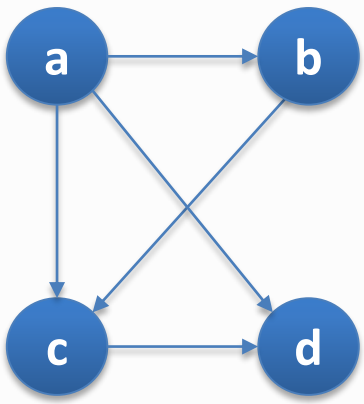
Shembull 12.7

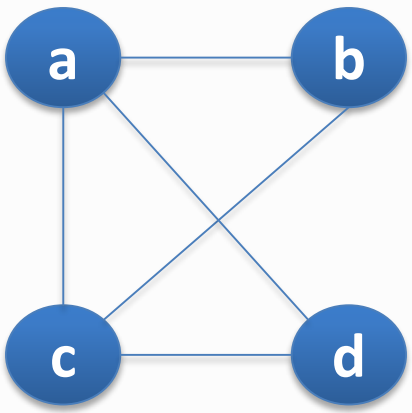
- Në këtë shembull do të tregohet algoritmi i Dijkstra-s.

```
61
62  int main()
63  {
64      int grafi[N][N] = {
65          { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
66          { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
67          { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
68          { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
69          { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
70          { 0, 0, 4, 0, 10, 0, 2, 0, 0 },
71          { 0, 0, 0, 14, 0, 2, 0, 1, 6 },
72          { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
73          { 0, 0, 2, 0, 0, 0, 6, 7, 0 }
74      };
75
76      cout << "Shtegu me i shkurter - Algoritmi i Dijkstra-s\n\n";
77      dijkstra(grafi, 0);
78      cout << endl;
79      system("Pause");
80      return 0;
81  }
```

Matrica e fqinjësisë

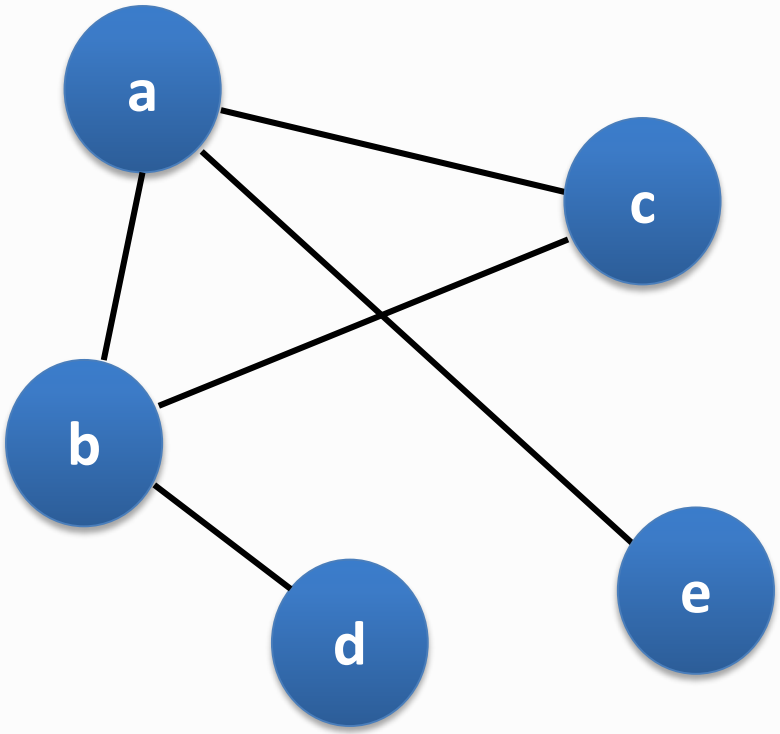
$$A[i,j] = a_{ij} = \begin{cases} 1 & \text{nëse } (i,j) \in E \\ 0 & \text{përndryshe} \end{cases}$$





	a	b	c	d
a	0	1	1	1
b	0	0	1	0
c	0	0	0	1
d	0	0	0	0

	a	b	c	d
a	0	1	1	1
b	1	0	1	0
c	1	1	0	1
d	1	0	1	0

Lista e fqinjësisë

a	b c e
b	a c d
c	a b
d	b
e	a