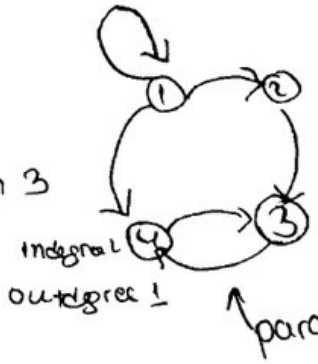
Graphs

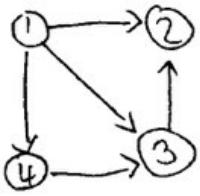
$G=(V,E)$, V - set of vertices , E - set of edges

1. Directed Graph

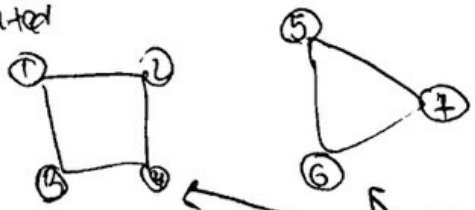


outgoing from 4 & incoming on 3
↓ it is incident from 4
and incident on 3

2. Simple Digraph - NO PARALLEL edges, no self-loop



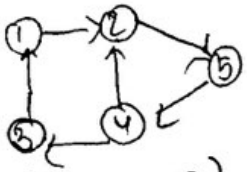
3. Non-connected



2 components

Ако имаме ребро от 4→6, тогава графът става свързан, но има 2 компоненти

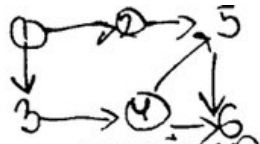
4. Strongly-connected - от всеки връх можеш да стигнеш до всеки друг.



Path - set of all vertices , (1→2→5)

cycle - 1→2→5→4→3→1

5. Directed Acyclic Graph (DAG)



Можем да подредим върховете му линейно, така че ребрата са само във forward direction

Topological ordering - ребрата са подредени в forward direction

Representation of Graph

Adj Matr.

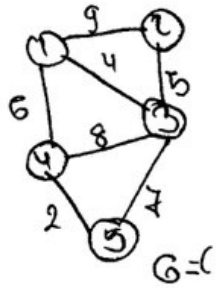
A	1	2	3	4	5
1	0	1	1	1	0
2	1	0	1	0	0
3	1	1	0	1	1
4	1	0	1	0	1
5	0	0	1	1	0

A[i][j]=1/0

n×n=n²

O(n²)

5×5

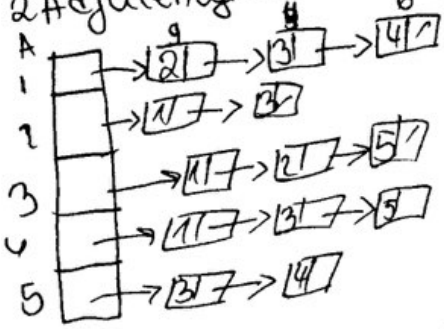


G=(V,E), |V|=n=5

|E|=e=7

1. Adjacency Matrix (матрица на съседство)
2. Adjacency list
3. Compact list

2 Adjacency List - Array of linked list



|V| + 2.|E| = n+2e

(реф. в друг 2 пъти)

Each location is representing a vertex

Ако имаме тегла вместо 1 и 0 можем да пишем теглата.

това става cost adjacency matrix

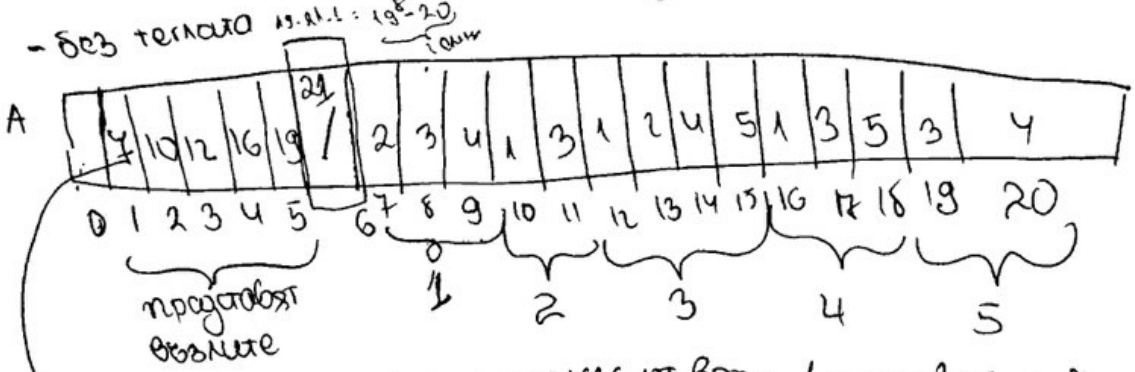
За 2. трябва заедно с името на върха, трябва да съхраняваме и теглото

3. Compact dist Representation - две линейни масива

$|V| + 2|E| + 1 = 5 + 2 \cdot 7 + 1 = 20$ за размера на масива

+1 защото започва от 1

- без теглата



Означава, че дъгите тръгват от върх 1 започват от 7 позиция нататък

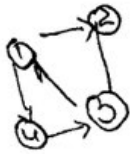
$|V| + |2|E| = n + 2e = 3n \approx O(n)$

Representation of Directed Graph

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

(1 2 3 4)

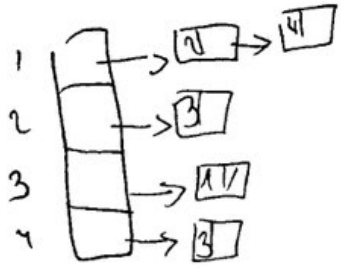
$n \times n = n^2$



$|V|$

$|E| = 5$

Adj. list



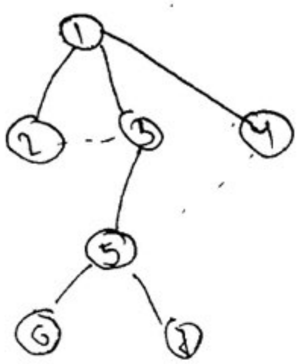
→ going out edges

$|V| + |E| = n + e = \approx \Theta(n)$

можем да имаме и ingoing edges

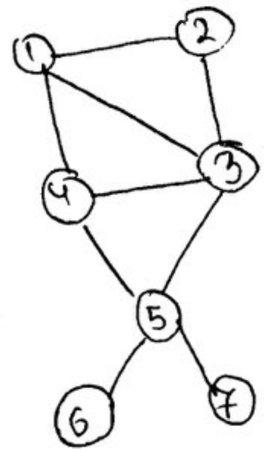
BFS - търсене в ширина

1. Similar to level order of Binary Tree



правя дърво от този граф

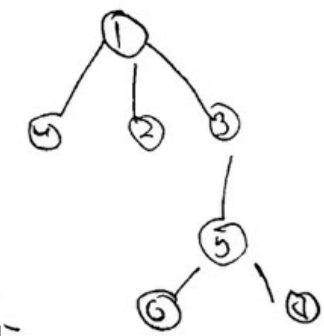
ако се получава цикъл, махаме ребро.



BFS: level ... | 1, 2, 3, 4, 5, 6, 7

BFS: 1, 4, 2, 3, 5, 6, 7

Не задължително да започнем от връх 1



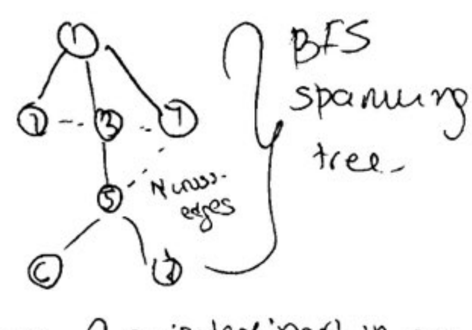
2. Когато посетим един връх, трябва да изучим (explore) всичките му adjacent vertices, можем да ги explore-ваме във всякакъв ред

Text Book Definition:

1. Visiting
2. Exploring - означава visiting all adjacent vertices of 4

За графа: Започв. от 1.

BFS: 1 visited 2, 3, 4, 5, 6, 7



Queue: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, ~~7~~

Махаме 1 от опашката и започваме explore. 2 e visited, insert in queue 3 и 4 и т.н. След като оп. е 2, започваме да explore 3

time: O(n)

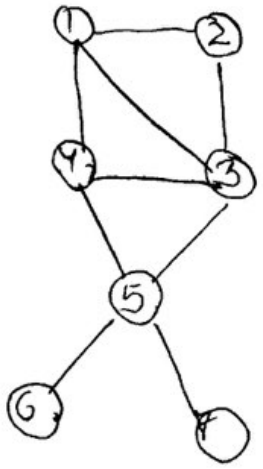
BFS - program

	0	1	2	3	4	5	6	7
0								
1		0	1	1	1	0	0	0
2		1	0	1	0	0	0	0
3		1	1	0	1	1	0	0
4		1	0	1	0	1	0	0
5		0	0	1	1	0	1	1
6		0	0	0	0	1	0	0
7		0	0	0	0	1	0	0

q: | 1 | 2 | 3 | 4 | | | |

visited: | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

0 1 2 3 4 5 6 7



visited е с 0 защото никой от върховете не е посещен първоначално

Всички са глобални тук.

```
void BFS (int i)//st. vertex
{
  int u;
  printf("%d", i);

  visited[i]=1

  enqueue(q,i); //push_back
  while(!isEmpty(q))
  {
    u=dequeue(q); // pop_front !
    for (v=1; v<=n; v++)
    {
      if (A[u][v]==1 && visited[v]=0)   // A[u][v]==1 - има ребро; visited[v]=0 - т.е. още не е посетен връх
      {
        printf("%d", v);
        visited[v]=1;
        enqueue(q,v);
      }
    }
  }
```

time: $O(n^2)$ //