**Дървета. Представяне на дърво. Покриващи дървета (MST)**

1. *Какво е дърво?*

*Дърво* наричаме **ацикличен граф**, съставен от една единствена компонента и съдържащ N-1 ребра.

То е разделено на **нива**, наричани *дълбочина* или *височина* на дървото. Върхът на *най-ниското* ниво се нарича **корен** – има *наследници*, но *няма родител*; върховете, които имат *родител*, но *не и* *наследници*, наричаме **листа**.

КОРЕН

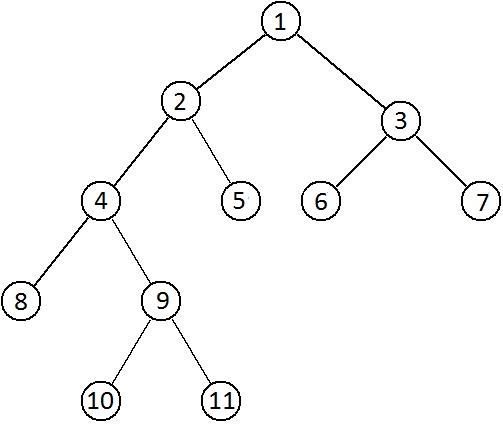
0

1

2

3

4



ЛИСТА

Родител на върха X е този връх, който се намира на ниво depth[X]-1 и има ребро от него до X. Всеки връх в дървото има само един родител (с изключение на корена).

Ребрата (клоните) на дървото могат да имат тегла, т.е*. дървото може да е претеглено.*

**P-ично дърво** значи от всеки връх да излизат *най-много* P на брой ребра. Показаното горе дърво е двоично – всеки връх има 2 наследника.

1. *Как да представим дърво в паметта на компютъра?*

Както вече казахме, дървото е частен случай на граф. Т.е. го представяме като такъв – със списък на съседите е най-удачният вариант.

Но понякога ни се налага от този граф да направим дърво. За неговото представяне използваме 2 масива: **tree**[] – пазим *родителя* на даден връх, и **depth** [] – пазим *дълбочината (нивото)*.

Най-лесно попълването става чрез алгоритмите за обхождане на граф.

* BFS

**void BFS(int v)**

**{**

**used[v]=1;**

**tree[v]=0;** //първият връх няма родител

**depth[v]=0;** //дълбочитана му е 0

**queue<int> q;**

**q.push(v);**

**int top;**

**while(!q.empty())**

**{**

**top=q.front();**

**for(int i=0; i<a[top].size(); i++)**

**{**

**if(used[a[top][i]]==0)**

**{**

**q.push(a[top][i]);**

**tree[a[top][i]]=v;** //родителят на всеки следващ връх е върхът, от който идваме

**depth[a[top][i]]=depth[v]+1;** //и дълбочината е с едно по-голяма от тази на родителя

**used[a[top][i]]=1;**

**}**

**}**

**q.pop();**

**}**

**}**

* DFS

**void DFS(int v, int p)**

**{**

**used[v]=1;**

**tree[v]=p;** //отбелязваме родителя

**depth[v]=depth[p]+1;** //и дълбочината

**for(int i=0; i<a[top].size(); i++)**

**if(used[a[top][i]]==0)**

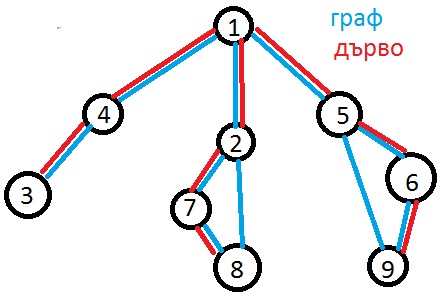
**dfs(a[top][i], v);**

**}**

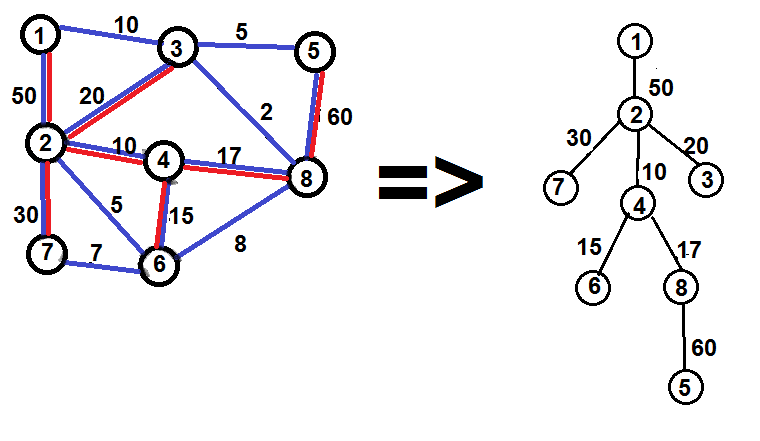
1. *MST (Min/Max Spanning Tree)*

**Минимално/максимално покриващо дърво** (наричано още *оптимално покриващо дърво*) е алгоритъм, който изгражда покриващо дърво върху *претеглен* граф, използвайки *максималните* (или *минималните*) ребра.

**Покриващо** дърво е това дърво, което включва всичките върхове на граф, съставен от една компонента.



Пример за максимално покриващо дърво:



Алгоритмите за построяване на MST са следните:

* Прим

**void prim(int root){**

**used[root]=1;**

**depth[root]=0;** //дълбочината на началния връх е 0

**for(int i=1; i<=n; i++){** //за всеки връх

**tree[i]=a[root][i];** //присвояваме теглото между него и първия

**p[i]=root;** //и добавяме за родител началния връх

**}**

**p[root]=0;** //родител на първия връх е 0

**tree[root]=INF;** //дължина от първия до първия връх не съществува

**int maxDist, j=0;**

**for(int l=0; l<n-1; l++){** //n-1 пъти се върти цикъла

**maxDist=0, j=-1;** //максималната дължина е 0 и е към -1 връх

**for(int i=1; i<=n; i++){** //за всеки връх

**if (tree[i]>maxDist && used[i]==0**){ //ако дължината му от първия е по-голяма от максималната и той е необходен

**maxDist=tree[i];** //присвояваме това ребро за максимално

**j=i;** //и отбелязваме кой е върха

**}**

**}**

**used[j]=1;** //последния отбелязан връх е обходен

**for(int i=1; i<=n; i++){** //за всеки връх

**if (tree[i]<a[j][i] && used[i]==0){** //ако съшествува по-голямо ребро до някой от другите върхове и той не е използван

**tree[i]=a[j][i];** //обновяваме списъка на максималните ребра

**p[i]=j;** //и родител на текущото ребро става отбелязаното

**depth[i]=depth[j]+1;** //а дълбочината му - това на педшественика +1

**}**

**}**

**}**

**for(int l=0; l<n/2; l++)**  //n/2 пъти

**for(int i=1; i<=n; i++)**

**depth[i]=depth[p[i]]+1;** //оптимизираме дълбочината на дървото

**}**

* Крускал

**int find\_set(int a){** //намира в кое множество се намира даден връх

**if(p[a]==a)return a;**

**else return find\_set(p[a]);**

**}**

**void union\_sets(int a, int b){**  //обединява две множества

**a=find\_set(a);**

**b=find\_set(b);**

**p[b]=a;**

**}**

**void kruskal(){**

**int a, b, l;**

**for(int i=0; i<m; i++){**

**a=v[i].second.first;** //извличаме двата върха и дължината на реброто

**b=v[i].second.second;**

**l=v[i].first;**

**if(find\_set(a)!=find\_set(b)){** //ако не са в едно множество

**cost+=l;** //добавяме дължината на реброто в общата цена

**union\_sets(a, b);** //обединяваме множествата

**res.push\_back(make\_pair(a, b));** //и добавяме реброто в минималното покриващо дърво

**}**

**}**

**}**