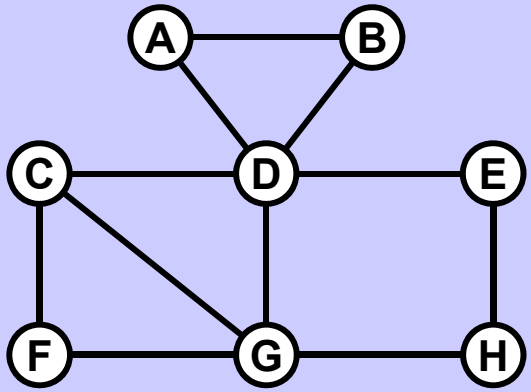
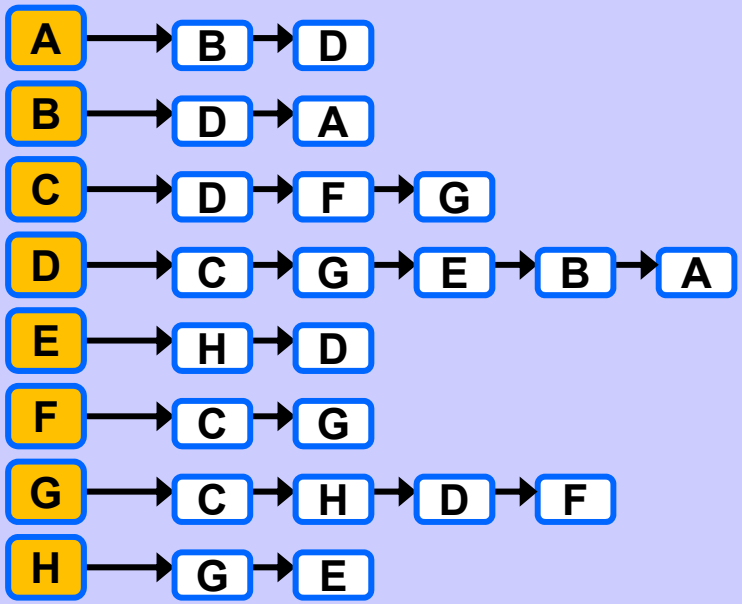
Průchod grafem do hloubky



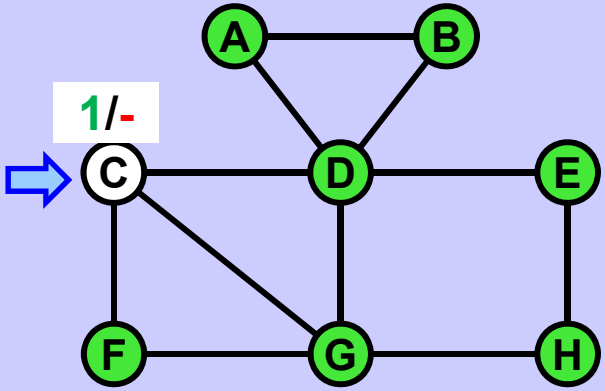
Spojová reprezentace



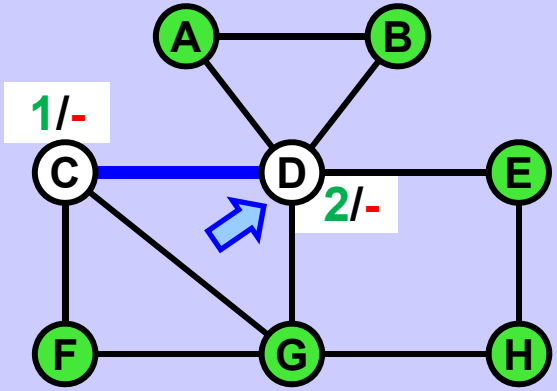
Matice sousednosti

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	0	0	0
B	1	0	0	1	0	0	0	0
C	0	0	0	1	0	1	1	0
D	1	1	1	0	1	0	1	0
E	0	0	0	1	0	0	0	1
F	0	0	1	0	0	0	1	0
G	0	0	1	1	0	1	0	1
H	0	0	0	0	1	0	1	0

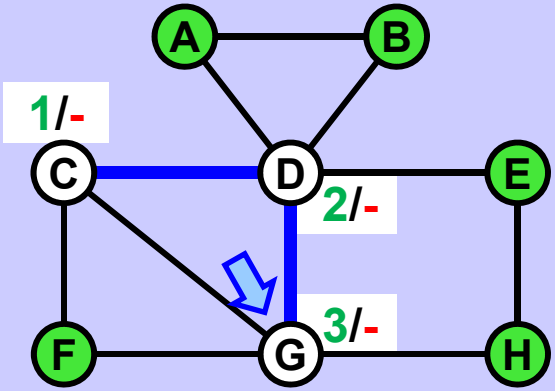
Průchod grafem do hloubky



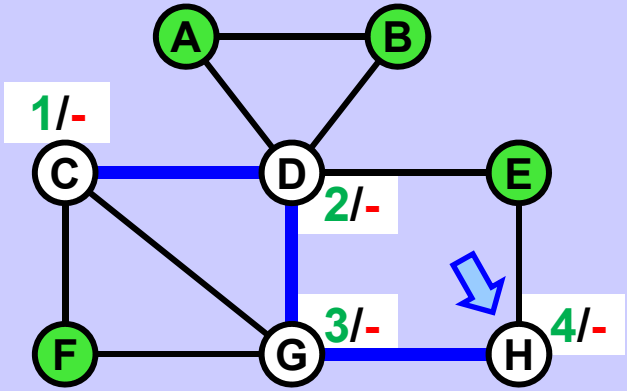
Zásobník	C
Výstup	C



Zásobník	C D
Výstup	C D

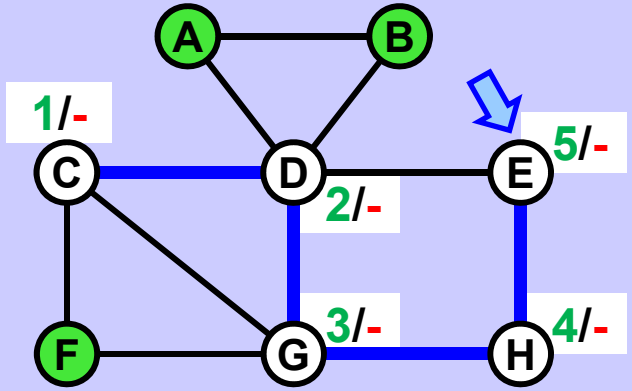


Zásobník	C D G
Výstup	C D G

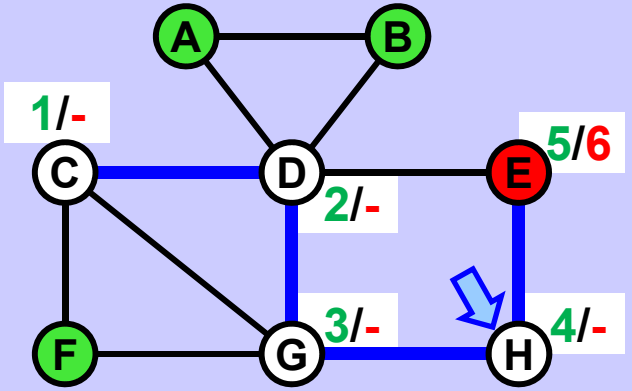


Zásobník	C D G H
Výstup	C D G H

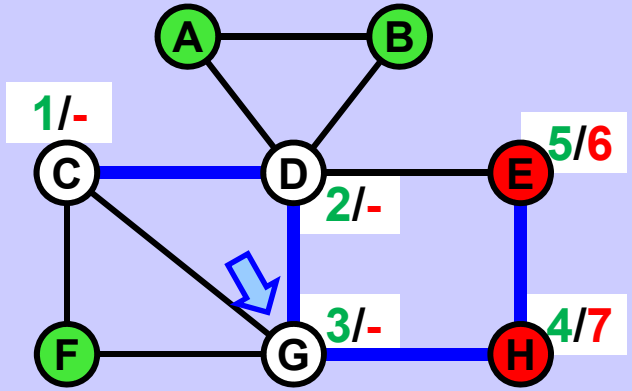
Průchod grafem do hloubky



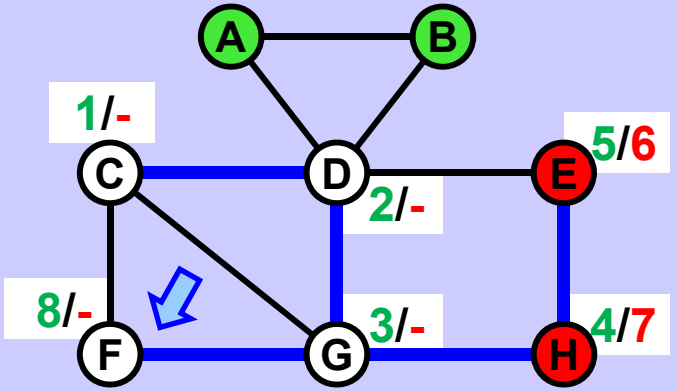
Zásobník	C D G H E
Výstup	C D G H E



Zásobník	C D G H
Výstup	C D G H E

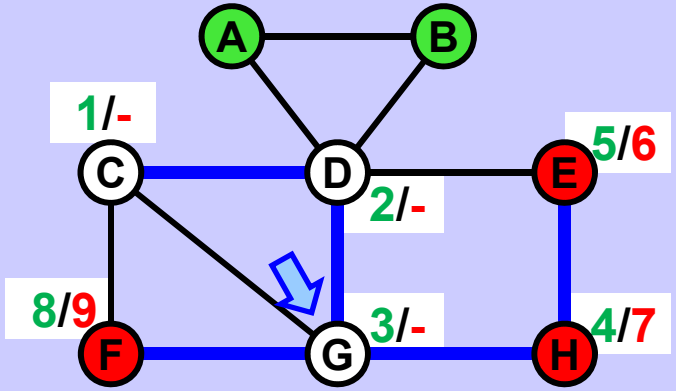


Zásobník	C D G
Výstup	C D G H E

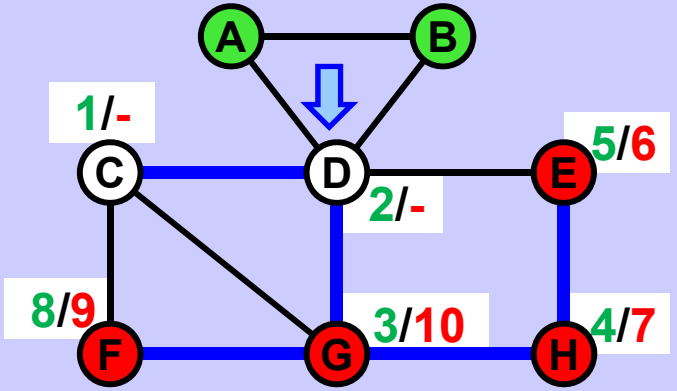


Zásobník	C D G F
Výstup	C D G H E F

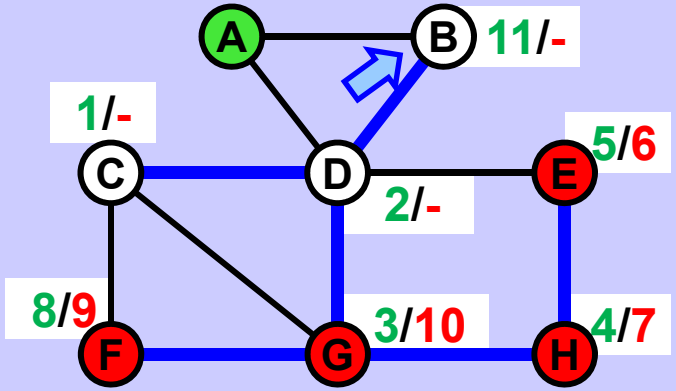
Průchod grafem do hloubky



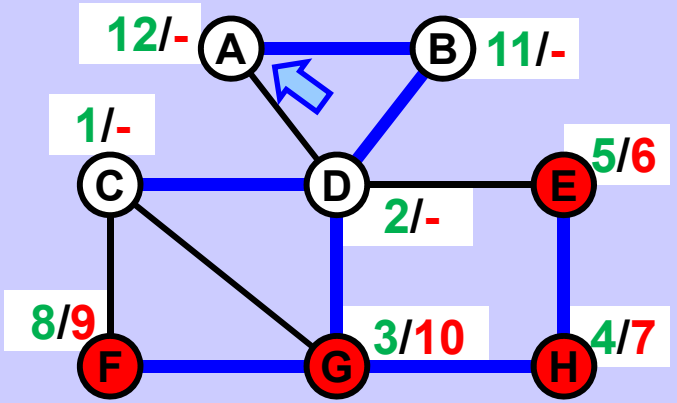
Zásobník	C D G
Výstup	C D G H E F



Zásobník	C D
Výstup	C D G H E F

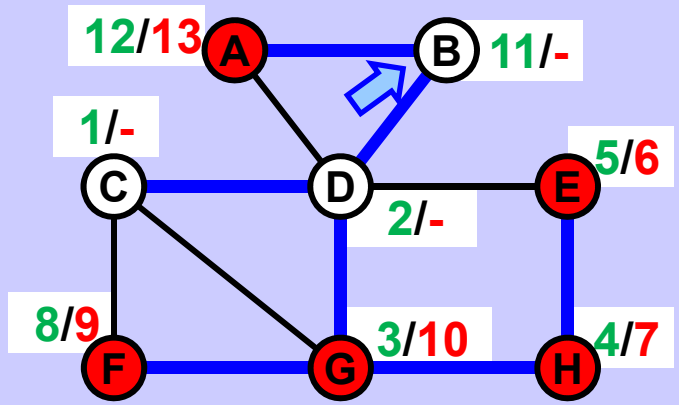


Zásobník	C D B
Výstup	C D G H E F B

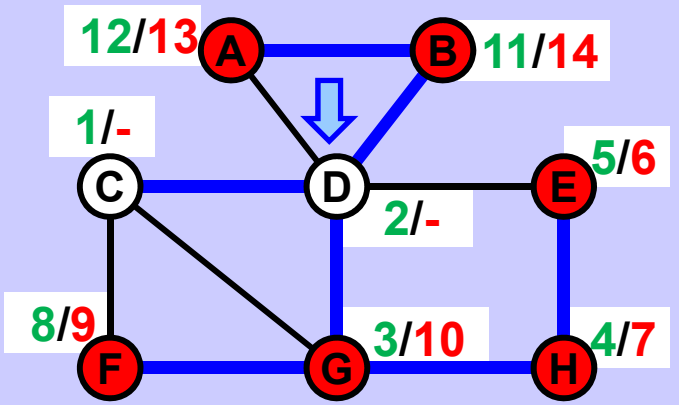


Zásobník	C D B A
Výstup	C D G H E F B A

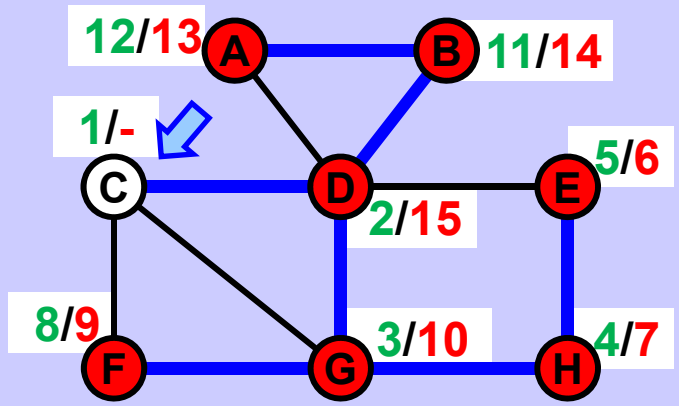
Průchod grafem do hloubky



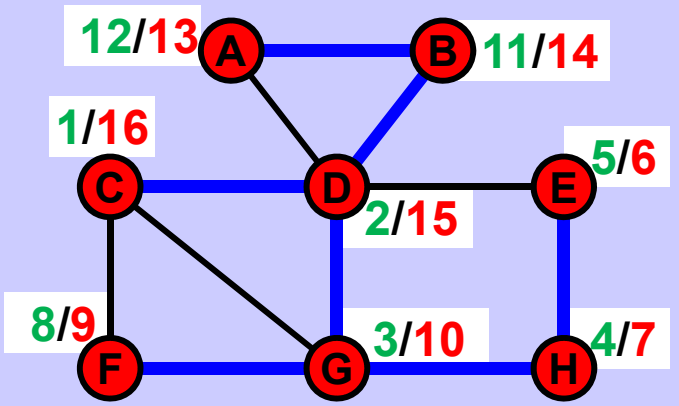
Zásobník	C D B
Výstup	C D G H E F B A



Zásobník	C D
Výstup	C D G H E F B A



Zásobník	C
Výstup	C D G H E F B A



Zásobník	
Výstup	C D G H E F B A

Průchod grafem do hloubky

Životní cyklus uzlu při prohledávání grafu
Fresh - open - closed (čerstvý - otevřený - uzavřený)

Fresh

Čerstvé uzly jsou všechny dosud ani jednou nenavštívené uzly.
Před začátkem prohledávání jsou všechny uzly čerstvé.
Při první návštěvě uzlu se uzel stává otevřeným.
Množina čerstvých uzlů se během prohledávání nikdy nezvětšuje vzhledem k inkluzi.

Open

Otevřené uzly jsou alespoň jednou navštívené uzly, které dosud nebyly uzavřeny.
Množina otevřených uzlů se může během prohledávání zvětšovat i zmenšovat.

Closed

Uzavřené uzly jsou uzly, které už během prohledávání nebudou navštíveny.
Pokud jsou všechny sousedy aktuálního uzlu otevřené nebo uzavřené, aktuální uzel se stává uzavřeným.
Množina uzavřených uzlů se během prohledávání nikdy nezmenšuje vzhledem k inkluzi.
Na konci prohledávání jsou všechny uzly uzavřené.

Průchod grafem do hloubky

Implementační poznámka

Fresh: Čerstvý uzel nemá přiřazen otevírací (ani zavírací) čas.
Open: Otevřený uzel nemá přiřazen zavírací čas.
Closed: Uzavřený uzel má přiřazen zavírací čas.

Otevírací a zavírací časy uzlů v některých případech prohledávání není nutno udržovat.

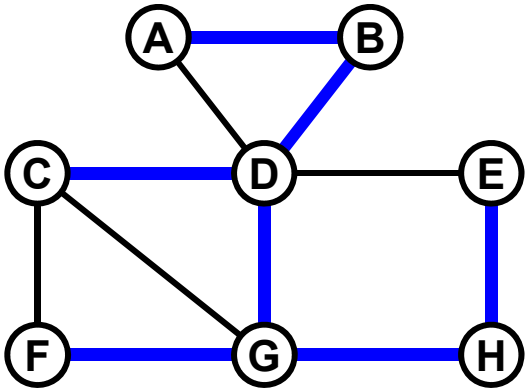
Při iterativním prohledávání s vlastním zásobníkem je ale nutno průběžně udržovat informaci u každého uzlu, zda je čerstvý, otevřený nebo zavřený.

V rekurzivním zpracování není nutno dělat explicitně ani to. Každé volání rekurzivní funkce odpovídá zpracování jednoho uzlu a všem jeho návštěvám. Při zavolání funkce se otevírá uzel, který je aktuálním parametrem volání, a na konci volání se tento uzel uzavírá. V těle funkce probíráme postupně sousedy aktuálního uzlu a voláme rekurzivně prohledávání pouze na ty z nich, které jsou ještě čerstvé (fresh). Stačí pak v každém uzlu udržovat jen informaci jednobitovou -- fresh nebo not fresh.

Průchod grafem do hloubky

Postupný obsah zásobníku

C
C D
C D G
C D G H
C D G H E
C D G H
C D G
C D G F
C D G
C D
C D B
C D B A
C D B
C D
C
--



Výpis (zpracování) uzlu při otevírání uzlu vede na posloupnost

C D G H E F B A

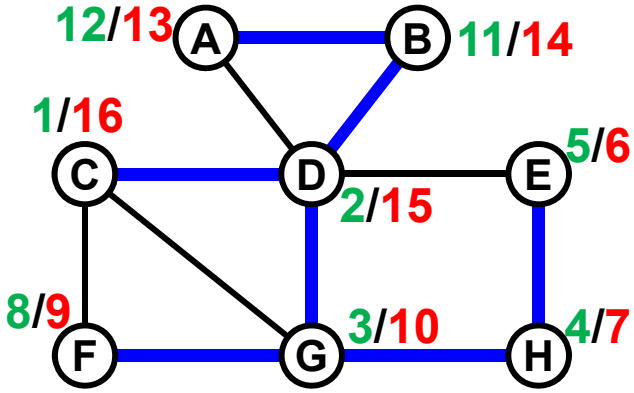
Výpis (zpracování) uzlu při zavírání uzlu vede na posloupnost

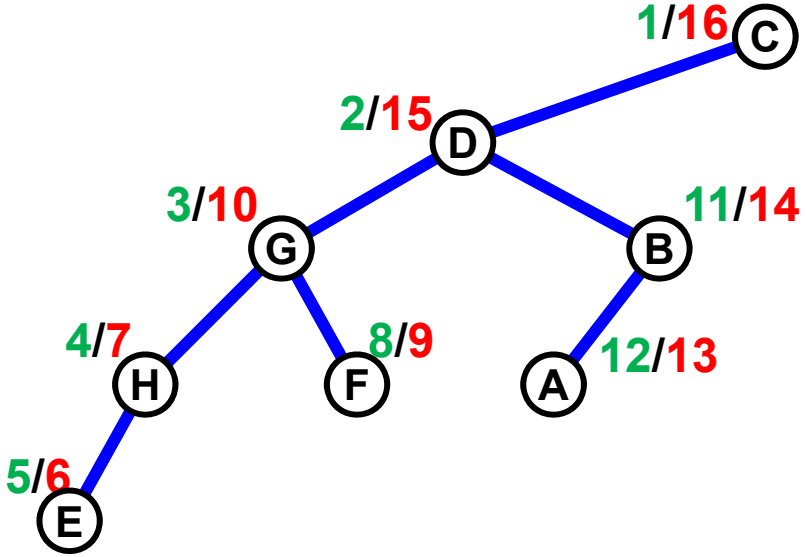
E H F G A B D C

Zpracování uzlu při jeho závírání se uplatní např. při hledání mostů nebo artikulací v neorientovaném grafu a při detekci silně souvislých komponent v orientovaném grafu.

Průchod grafem do hloubky

Strom průchodu do hloubky s otevíracími a zavíracími časy jednotlivých uzlů





Všimněme si, že v podstromu s kořenem X pro každý uzel Y různý od X platí
Open_time(X) < Open_time(Y) < Close_time(Y) < Close_time(X).
Naopak, pokud Y neleží v podstromu s kořenem X, pak platí
Close_time(X) < Open_time(Y) nebo Close_time(Y) < Open_time(X)

Počet uzlů v podstromu s kořenem X je pak
(Close_time(X) + 1 – Open_time(X)) / 2.

Průchod grafem do hloubky - Python

```
def DFS( graph ):
    visited = [False] * graph.size
    stack = Stack()
    stack.push( graph.nodes[0] ) # start search in node 0
    visited[0] = True
    while not stack.isEmpty():
        node = stack.pop()
        print(node.id, end = " ") # process the node
        for neigh in node.neighbours:
            if not visited[neigh.id]:
                stack.push( neigh )
                visited[neigh.id] = True
```

Průchod grafem do hloubky rekurzivně - Python

```
def DFSrec( node, visited ):
    visited[node.id] = True
    print( node.id, end = " " )   # process the node
    for neigh in node.neighbours:
        if visited[neigh.id] == False:
            DFSrec( neigh, visited )

def DFSrecRun( graph ):
    visited =  [False] * graph.size
    DFSrec( graph.nodes[0], visited )
```

Průchod grafem do hloubky Cpp

```
void DFS_rec_full( int start ) {
  vector<int> openT( N, 0 );
  vector<int> closeT( N, 0 );
  vector<int> pred( N, -1 );  // -1 == no predecessor
  int time = 0;
  DFS_rec_full( start, time, openT, closeT, pred );

  for( int n = 0; n < N; n++ )
    cout << "node " << n << " open/close "
         << openT[n] << "/"<< closeT[n]
         << " pred " << pred[n] << endl;
}
```

Průchod grafem do hloubky Cpp rekurzivně

```
void DFS_rec_full( int currNode, int & time,
                   vector<int> & openT,
                   vector<int> & closeT,
                   vector<int> & pred ){

  int neigh;
  openT[currNode] = ++time;
  for( int j = 0; j < edge[currNode].size(); j++ ){
    neigh = edge[currNode][j];
    if( openT[neigh] == 0 ) {
      pred[neigh] = currNode;
      DFS_rec_full( neigh, time, openT, closeT, pred );
      cout << currNode << " --> " << neigh << endl;
    }
  }
  closeT[currNode] = ++time;
}
```

Průchod grafem do hloubky rekurzivně, základní varianta

```
// no time stamps
void DFS_rec_plain( int currNode, vector<bool> & fresh ){
  int neigh;
  fresh[currNode] = false;
  for( int j = 0; j < edge[currNode].size(); j++ ){
    neigh = edge[currNode][j];
    if( fresh[neigh] ) {
      DFS_rec_plain( neigh, fresh );
      cout << currNode << " --> " << neigh << endl;
    }
  }
}

void DFS_rec_plain( int start) {
  vector<bool> fresh( N, true );
  DFS_rec_plain(start, fresh);
}
```

Průchod grafem do hloubky

Asymptotická složitost

Každá jednotlivá operace se zásobníkem a s použitými datovými strukturami má konstantní složitost pro jeden uzel (včetně inicializace).

Každý uzel jen jednou vstoupí do zásobníku a jednou z něho vystoupí.

Stav uzlu (fresh/open/closed) se testuje tolikrát, kolik je stupeň tohoto uzlu, přičemž jeden test proběhne také v konstantním čase. Součet stupnů všech uzlů je roven dvojnásobku počtu hran.

Celkem tedy

$$\Theta(|V| + |E|).$$