Implementácia údajovej štruktúry strom

Znenie zadania:

Pridajte podporu operácie PrintRow, ktorá vypíše hodnoty vrcholov na zadanej úrovni stromu.

Prototyp operácie PrintRow:

```
void PrintRow ( Tree T, int Level );
```

Pridajte podporu operácie PrintSubtree, ktorá vypíše hodnoty vrcholov postromu, ktorého koreňom je prvok so zadaným indexom.

Prototyp operácie PrintSubtree:

```
void PrintSubtree( Tree T, int Index );
```

Poznámka: Pre overenie činnosti operácie doplňte aj modul pre testovanie.

Opis riešenia:

Funkcia PrintRow najprv zistí koľko prvkov sa bude vypisovať. Táto hodnota je zároveň aj hodnota prvého prvku v danej úrovni. Nasleduje cyklus v ktorom sa vypíšu postupne všetky prvky v úrovni.

Funkcia printSubtree funguje podobne ako funkcia PrintTree. Rozdiel je ale v tom že sa nezačína od prvej úrovne ale od n tej úrovne. Tá sa najprv vypočíta pomocou cyklu. Je ešte potrebné pri každej úrovni vypočítavať ktorý je prvý vypisovaný prvok. Ten sa počíta vždy ako dvojnásobok prvého prvku z predošlej úrovne. Potom už iba nasleduje cyklus, ktorý vypíše jednotlivé prvky rovnako ako pri PirntTree.

Modul pre testovanie nebolo potrebné nijak upravovať, iba dopísať volanie funkcií.

Zdrojový kód:

funkcia PrintRow(Tree T, int Level):

```
void PrintRow( Tree T, int Level ){
    if(T == NULL) {FatalError("No tree!");}else{
        if (Level <= T->MaxLevel){

            int i, k;
            k=pow(2,Level-1);
            for(i=0; i<pow(2,Level-1); i++)
             {
                 printf("%02d ", T->Array[k++]);
             }
            putchar('\n');
        }
    }
}
```

funkcia void PrintSubtree(Tree T, int Index):

```
void PrintSubtree( Tree T, int Index ){
    if(T == NULL) {FatalError("No tree!");}else{
        if (Index <= pow(2,T->MaxLevel)){

        int i = 1, j, k = Index, l, m, n = 0, temp = k;

        do n++; while (k>=pow(2,n));

        m = 4*pow(2,T->MaxLevel-(n));
```

```
            for (n; n<=T->MaxLevel; n++){
                k = temp;
                for (j = 1; j<=i; j++){
                    l = (m-2*i)/i;
                     if(j==1) PrintSpaces(l-(l/2)); else PrintSpaces(l);
                    printf("%02d", T->Array[k++]);
                }
                temp*=2;
                i*=2;
                putchar('\n');
            }

            }

        }
    }
```

Implementácia údajovej štruktúry graf

Znenie zadania:

Nájdite chybu a opravte ju v implementácii operácie pre prehľadávanie grafu do hĺbky dfs(). Funkcia dfs() pre svoju činnosť využíva rekurzívnu funkciu dfsr().

Pridajte podporu operácie dfsst(), ktorá vypíše hrany tvoriace kostru grafu (spanning tree) získanú prehľadávaním do hĺbky. V prípade, že zadaný graf nie je súvislý, vypíšte všetky kostry (jeho súvislých komponentov) získané prehľadávaním do hĺbky (spanning forest).

Prototyp operácie dfsst:

```
void dfsst(Graph G);
```

Poznámka: Pre overenie činnosti operácie doplňte aj modul pre testovanie.

Opis riešenia:

Funkcia dfsr má postupne prechádzať jednotlivé prvky grafu. Každý prvok ktorý navštívi treba označiť. To sa však v pôvodnej implentácii neudialo. Preto bolo potrebné pridať označenie navštíveného uzla.

Funkcia dfsst funguje rovnako ako funkcia dfs. Tá prechádza všetky prvky grafu a postupne volá funkciu PrintEdge, ktorá je ekvivalentom k dfsr, akurát výpis je potrebné robiť iba keď sa nájde hrana medzi prvkami grafu. Jej existencia sa dokáže ak je v matici 1. Aby sa predišlo duplicitnému hľadaniu hrán sa preto v podmienke porovnáva aj či už daný prvok na ktorý ukazuje hrana nebol navštívený.

Modul pre testovanie opäť nepotreboval skoro žiadne úpravy, len doplnenie volania funkcie dfsst.

Zdrojový kód:

funkcia void dfsr (int X, Graph G):

```
void dfsr(int x, Graph G)
{
   int j;
   G->visited[x]=1;
   printf("The node visited: %d\n",x);
   for(j=0;j<G->nodes;j++)
      if(G->adj[x][j] ==1 && G->visited[j] ==0)
           dfsr(j,G);
}
```

funkcia void Edge(int X, Graph G):

```
void printEdge(int x, Graph G)
{
   int j;
   G->visited[x]=1;
   for(j=0;j<G->nodes;j++)
      if(G->adj[x][j] ==1 && G->visited[j] ==0){
          printf("Edge: (%d,%d)\n",x,j);
          printEdge(j,G);
       }
}
```

funkcia dfsst(Graph G):

```
void dfsst( Graph G){
     int n;
     ClearVisited(G);


     for(n=0; n<G->nodes; n++){
         if(G->visited[n] ==0){
             printEdge(n,G);
         }
     }
}
```