АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА ЕКСКУРЗИЯ

1. Алгоритъм за намиране на всички прости пътища

Ако изпълним познатото ни обхождане в ширина или дълбочина, започвайки от *i*, ще намерим *един* възможен прост път от *i* до *j*, тъй като задължително ще преминем през всички върхове от същата компонента (в това число и през върха *j*). Естествено, ако двата върха не принадлежат на една и съща компонента на свързаност, то път между тях не съществува и такъв няма да бъде намерен. Модификацията на алгоритъма за обхождане в дълбочина се състои в последователното рекурсивно изпълнение на *DFS*(*k*) за *всеки* връх *k*, съседен на текущо разглеждания връх *i*, а не само за съседа с минимален номер. Така основният цикъл в програмата няма да се промени:

```
for (k = 0; k < n; k++)
if (A[i][k] && !used[k]) allDFS(k, j);
```

Разликата ще бъде, че в новата реализация ще извършим присвояването used[i] = 0 след връщане от рекурсията, докато при обикновеното *DFS* това го нямаше. В примера от *фигура 5.3.1.* това означава следното: Нека в момента разглеждаме връх *2* — той има четири съседа: *1*, *4*, *3*, *5*. От връх *1* сме дошли и затова няма да тръгваме по него. Изпълнявайки for-цикъла, избираме връх *3* и стартираме *DFS*(*3*). Така рекурсивно ще продължим обхождането по-нататък от връх *3*, но след връщане от рекурсията в края на *DFS*(*3*) ще се изпълни присвояването used[3] = 0. По-нататък, когато обхождането продължи със следващите съседи — върховете *4* и *5*, при тези обхождания ще бъде възможно *отново* да се премине през връх *3*. Това не е така при обикновеното *DFS* — при него веднъж маркираме ли връх *3* с used[3] = 1, той вече не е достъпен на никоя следваща стъпка от обхождането.
Ще въведем и масив path[], в който ще пазим върховете по реда на тяхното обхождане. Така, ако на някоя стъпка достигнем до крайния връх-цел *j*, можем да отпечатаме текущия път. (По Преслав Наков).

2. Организация на програмата:

Матрицата на съседство е int A[MAXN][MAXN] → в нея съхраняваме връзките между селата и разстоянията между тях.
Структурата за съхраняване на всеки един намерен път:

```
struct verses{
 int dist; // Дължина на пътя
 int lnh;  // Брой върхове от които е съставен намерения път
 char rpnt[MAXN]; // Масив, който съдържа номерата на
върховете
}res[1000];
```

С помощта на функция **findPaths** намираме всички пътища, по-малки от лимита (променлива - **maxd**) и ги съхраняваме в масива от структури: **res[]**. Променливата **npaths** съдържа броя на пътищата, които удовлетворяват условието.
Остава сортировката и извеждането на резултата.

Ползваме функция **qsort**:

```
qsort(res, npaths,sizeof(verses),cmp);
```

Функцията за сравнение cmp:

```
int cmp(const void *a,const void *b)
```

```
{
  verses *c, *d;
  c=(verses *)a;
  d=(verses *)b;
  if(c->dist > d->dist) return 1;
  if(c->dist < d->dist) return -1;
  return(strcmp((*c).rpnt,(*d).rpnt));
}
```

Пояснение:
Ако има няколко маршрута с еднакви дължини, те се подреждат във възходящ ред на номерата на населените места, които формират съответните маршрути. Например: При еднакви дължини, маршрута <1 10 12 11 8> ще предхожда маршрута <1 10 12 20 14 8>, защото населено място №11 предхожда населено място №20.

Автор: Пано Панов