АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА ТЕЛЕФОН 112

Ще представим авторовото решение на трите задачи едновременно, тъй като задачата **no122C** е подзадача на **no122B**, а задачата **no122B** – подзадача на **no122A**.

В **no122C** се иска по зададен граф $G(V,E)$, $|V|=n$, $|E|=m$, да се отговаря на множество въпроси от вида „Има ли път от връх u до връх v?“. Такъв въпрос е еквивалентен на въпроса „Връх u и връх v в една и съща свързана компонента на графа ли са?“. За целта, най-добре е да се извърши едно обхождане на графа (все едно, в ширина или дълбочина) за време $O(m)$, при което да се идентифицират свързаните компоненти и всеки връх да се отбележи с идентификатора на свързаната компонента, на която принадлежи (например началния връх на обхождането на компонентата). Тогава проверката дали два зададени върха са в една и съща свързана компонента става за константно време и алгоритъмът е със сложност $O(m + Q)$, където Q е броят на обажданията на граждани.

В **no122B**, освен въпросите на гражданите за наличие на пътища между зададени върхове се добавят и обажданията на пътните служби за почистени пътища. Заявките на гражданите обработваме по същия начин както в задача **no122C**. За бързо обработване на съобщенията на пътните служби е добре да се използва абстрактния тип *разбиване*, пополярен още като *find-join-set* или като *find-merge-set*. В зависимост от начина на импелементиране на разбиването, ще получим и различни по бързина алгоритми. Най-добрата известна имплементация на разбиване е с дървета, при което операцията `join` се изпълнява за константно време, а операцията `find`, имплементирана с *повдигане* на дърветата и *балансиране* по височина, се изпълнява за време $O(\log^* n)$, което при малките стойности на n в задачата е равносилно на константа ($\log^* 1000 < 4$). Така получаваме и в този случай алгоритъм със сложност $O(m + Q)$. В предложеното решение не е използвано балансиране на дърветата по височина, тъй като при началното обхождане на графа *find-join*-поддърветата на всички свързани компоненти са с височина 1 и това е напълно достатъчно за получаването на бърз алгоритъм.

В задачата **no122A**, към двата вида заявки се добавя и трети вид – съобщенията за отпадане на някои от проходимите пътни отсечки между два града. При условие че вече имаме функция за обхождане на свързаните компоненти и маркиране всички върховете на обходената компонента с номера на началния връх, естественото решение на тази поздазача е при зададено ребро (u, v), което е станало непроходимо, да елиминираме това ребро и направим едно обхождане с начален връх u. Ако при това обхождане достигнем до v, значи отстраняването на реброто не е нарушило свързаността на компонентата. В противен случай правим още едно обхождане с начален връх v за да намерим върховете, които са останали в компонентата на v. Ако означим с Q' броя на заявките от първи и втори вид, а с Q'' – тези от трети вид, сложността на този алгоритъм в най-лошия случай ще бъде $O(mQ'' + Q')$.

Ето и решението на задача **no122A**, което съдържа в себе си решенията на останалите две задачи:

```
#include <stdio.h>
#define MAXN 1001

int U[MAXN+1],G[MAXN][MAXN],N,M;
int Q[MAXN],b,e;
void make_empty(){b=0;e=-1;}
```

```
void push(int x){Q[++e]=x;}
int pop(){return Q[b++];}
bool is_empty(){return b>e;}

void BFS(int r) {
  int  x,y,i;
  make_empty();push(r);U[r]=r;
  while(!is_empty())
  {  x=pop();
     for(i=1;i<=G[x][0];i++)
     {  y=G[x][i];
        if(!U[y]) {push(y);U[y]=r;}
     }
  }
}
int find(int x)
{   int y=x;
    while(U[y]!=y) y=U[y];
    while(x!=y){int z=U[x];U[x]=y;x=z;}
    return y;
}
void join(int x,int y)
{  U[x]=y; }

int main()
{   int i,j,k,q,R;
    scanf("%d %d",&N,&M);
    for(i=1;i<=N;i++) {G[i][0]=U[0]=0;}
    for(i=1;i<=M;i++)
    {  scanf("%d %d",&j,&k);
       G[j][++G[j][0]]=k;
       G[k][++G[k][0]]=j;
    }
    for(i=1;i<=N;i++)
       if(U[i]==0) BFS(i);
    scanf("%d",&R);
    for(i=1;i<=R;i++)
    {  scanf("%d %d %d",&q,&j,&k);
       switch (q) {
         case 1: if(find(j)==find(k)) printf("1");
                 else printf("0");
                 break;
         case 2: j=find(j); k=find(k);
                 if(j!=k) join(j,k);
                 break;
         case 3: if(U[j]=j){int t=j;j=k;k=t;}
                 BFS(j);if(U[k]!=j) BFS(k);
       }
    }
    printf("\n");
    return 0;
}
```

Автор: Красимир Манев