АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
РЕКОНСТРУКЦИЯ

Станцията може да бъде представена като неориентиран граф G с върхове $V = \{1, 2,..., N\}$ и цели положителни тегла на ребрата. Съгласно условието, G е свързан и между всеки два върха има единствен път. Следователно G е дърво. Задачата е по зададени дължини на пътищата между всеки два върха на дърво да се намерят двойките от върхове, които са свързани с ребра.

Да означим с $d(v,w)$ дължината на пътя от връх v до връх w в G. Ще наричаме $d(v,w)$ *разстояние* от v до w. Важно е да се отбележи, че разстоянието $d(v,w)$ между върховете на произволен свързан неориентиран граф, дефинирано като дължината на най-късия път между всеки два върха в графа, има същите свойства, както всяка друга мярка в математиката, наричана разстояние (например, разстоянието между две точки на Евклидовата равнина):

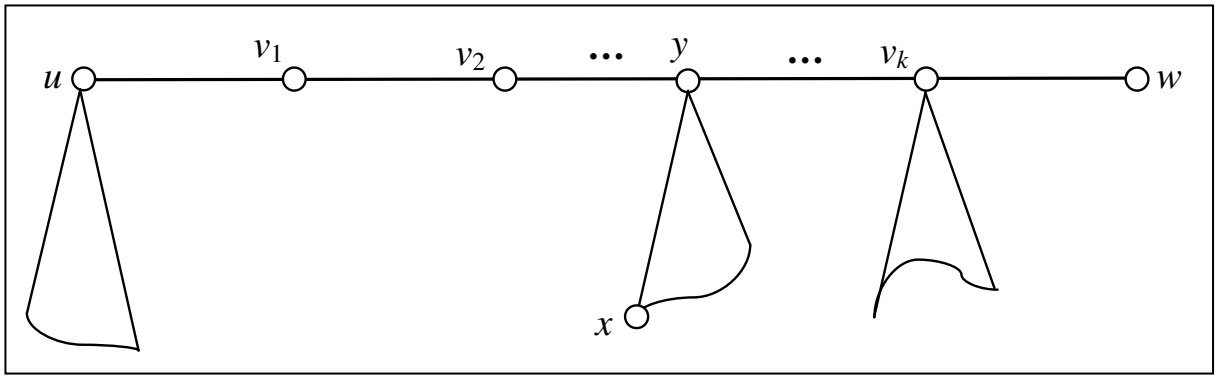
1. $d(v,w) \geq 0\ \forall v,w \in V$, като $d(v,w) = 0 <=> v = w$;
2. $d(v,w) = d(w,v,)\ \forall v,w \in V$;
3 $d(u,v) + d(v,w) \geq d(u,w)$ (неравенство на триъгълника)

Затова при решаване на задачи с разстояния в графи, можем да си представяме върховете като точки в Евклидовата равнина, като внимаваме да не свързваме разстоянието между две точки с отсечката между тях. Така например, докато в Евклидова равнина неравенството на триъгълника е равенство тогава и само тогава, когато точката v лежи на отсечката от u до w, то в графи неравенството на триъгълника е равенство тогава и само тогава, когато върхът v лежи на най-късия път от u до w.

Първият въпрос, който трябва да си зададем преди да започнем да решаваме задачата е: дали разстоянията между върховете в дърво определят еднозначно дървото. Ще докажем, че това е така, а разсъжденията от доказателството до голяма степен определят алгоритъма, който решава задачата.

Нека v и w са два върха от дървото (в решението дадено по-долу, при всеки избор на път сме търсили такива два върха, че разстоянието между тях да е максимално в G, но това не е съществено за доказателството). Съгласно казаното по-горе, върховете v, които лежат на пътя от u и w се определят еднозначно от равенството $d(u, v) + d(v, w) = d(u, w)$. Нека сортираме така намерените върхове в нарастващ ред на разстоянието им до u. Така възстановяваме разглеждания път u, v_1, v_2,…, v_k, w и ребрата участващи в него.

Пътят от u до w еднозначно определя множество от коренови поддървета на G, всяко едно от които има за корен някой от върховете u, v_1, v_2,…, v_k или w – като различните поддървета имат различни корени (виж Фигурата).



Възстановяването на останалата част на дървото ще сведем до възстановяване на всеки от получените подграфи. Да разгледаме произволен връх x, който не участва в пътя от u до w. Коренът y на съдържащото го поддърво определяме като решим спрямо низвестните $d(u,y)$, $d(y,w)$ и $d(y,x)$ системата от уравнения:

$$d(u,y) + d(y,w) = d(u,w)$$
$$d(u,y) + d(y,x) = d(u,x)$$
$$d(y,w) + d(y,x) = d(w,x)$$

За реконструирането на поддървото с корен y е достатъчно да разгледаме пътя от y до x и да извършим с него по-горните разсъждения и т.н. рекурсивно, докато получим път, върховете на който не са корени на поддървета. Алогично постъпваме и с останалите поддървета на пътя от u до w.

Направеният анализ подсказва съответен алгоритъм по схемата "разделяй и владей" и ни насочва към съответна рекурсивна процедура. Всъщност рекурсия в чист вид не е необходима, тъй като горните нива на рекурсията не използват резултати от долните нива. Изложената по-горе идея сме реализирали, като поставяме всяка нова подзадача в края на опашката `zarec`, а всеки път вземаме нова подзадача за решаване от началото на опашката. Получаваме следната програма:

```
#include <stdio.h>
#define MAXN 1025

int N,a[MAXN][MAXN],gr[MAXN][MAXN],used[MAXN], chain[MAXN];
int fromch[MAXN][2],zarec[MAXN][3],recb,rece;

void rec()
{  int c,i,j,x,y,z,tmp,pos,from,to,l;
   from=zarec[recb][0];to=zarec[recb][1];l=zarec[recb++][2];
   chain[0]=from;  c=1;
   for(i=1;i<=N;i++)
      if(i!=from&&i!=to)
         if(a[from][i]+a[i][to]==l)
         {   chain[c++]=i; used[i]=1; }
   chain[c]=to;
   if(c==1)gr[from][to]=gr[to][from]=used[from]=used[to]=1;
   else
   {  for(i=c-2;i>=0;i--)
         for(j=1;j<=i;j++)
         {  x=chain[j];y=chain[j+1];
            if(a[from][x]>a[from][y])
            { tmp=chain[j];chain[j]=chain[j+1];chain[j+1]=tmp;}
         }
      gr[from][chain[1]]=gr[chain[1]][from]=1;
      for(i=1;i<c-1;i++)
      {  x=chain[i];y=chain[i+1];
         gr[x][y]=gr[y][x]=1;
         used[x]=used[y]=1;
      }
      gr[chain[c-1]][to]=gr[to][chain[c-1]]=1;
      used[from]=1; used[to]=1;
   }

   for(i=0;i<=c;i++) fromch[i][0]=fromch[i][1]=0;
   for(pos=1;pos<=N;pos++)
   {  if(used[pos]) continue;
      else
      {  x=a[from][pos];y=a[pos][to];
         z=(x+l-y)/2;i=0;
         while(a[from][chain[i]]!=z) i++;
         if((x+y-l)/2>fromch[i][1])
```

```
         {  fromch[i][1]=(x+y-l)/2;fromch[i][0]=pos; }
      }
   }
   for(i=0;i<=c;i++)
      if(fromch[i][1]!=0)
         {  zarec[++rece][0]=chain[i];
            zarec[rece][1]=fromch[i][0]; used[fromch[i][0]]=1;
            zarec[rece][2]=fromch[i][1];
         }
   return;
}


int main()
{   int i,j,from,to,max=0;
    scanf("%d",&N);
    for(i=1;i<N;i++)
    {  for(j=i+1;j<=N;j++)
       {  scanf("%d",&a[i][j]); a[j][i]=a[i][j];
          if(a[i][j]>max){max=a[i][j];from=i;to=j;}
       }
    }
    for(i=1;i<=N;i++)
    {  used[i]=0;gr[i][0]=0;
       for(j=1;j<=N;j++) gr[i][j]=0;
    }
    recb=rece=0;
    zarec[0][0]=from;zarec[0][1]=to;zarec[0][2]=max;
    while (recb<=rece) rec();
    for(i=1;i<=N;i++)
    {   gr[i][0]=0;
        for(j=1;j<=N;j++) if(gr[i][j]) gr[i][0]++;
    }
    for(i=1;i<=N;i++)
    {   printf("%d",gr[i][0]);
        for(j=1;j<=N;j++)
           if(gr[i][j]) printf(" %d",j);
        printf("\n");
    }
    return 0;
}
```

При оценяване на сложността на този алгоритъм трябва да постъпим внимателно. Тъй като размерът M на входните данни е $O(N^2)$, ще търсим сложността на алоритъма не като функция на N, а на M. Така за въвеждане на данните и извеждане на резултата ще е необфодимо време $O(M)$. Втората особеност е, че при анализа на сложността важен елемент е броят на пътищата които алгоритъмът ще разгледа. Да означим този брой с $P(N)$. Той не може да надхвърля броя $N - 1$ на ребрата на дървото, тъй като всеки нов път идентифицира поне едно неидентифицирано до момента ребро. С всеки разглеждан път алгоритъмът извършва 2 съществени дейности – първо идентифицира ребрата, участващи в пътя и после избира за всяко от вкоренените в пътя поддървета връх от поддървото, с който да продължи работата. Всяка от дейностите изисква преглеждане на всички не участвали до момента в пътища върхове и значи всяка от тях има сложност $O(P(N).N)$. Тъй като $P(N)$ е $O(N)$, същинската част на алгоритъма е със сложност $O(N^2)= O(M)$. Затова използваният алгоритъм е със сложност $O(M)$.

Автор: Красимир Манев