Варианти на алгоритми. Влияние върху производителността. Въведение в анализа

алгоритмъм ← → структури данни

- **примерна задача: свързаност на обекти.**

(връзки (директни и опосредствени) в мрежа; точки в ел. м,режа, компоненти в чип; синонимни обръщения в програмирането)

Кога 2 точки са свързани? (не целим изброяване на всички пътища)

Теория на графи: "N обекта са свързани ако има точно N – 1 изведени двойки в алгоритъма на свързаност . [понятия: граф и опорно дърво].

вход	изход	
3 – 4	3 – 4	абстрактни операции, необх. за алгоритъма
4 -9	4 – 9	
8 – 0	8 – 0	*намиране множества от свързани ел.,
2 – 3	2 – 3	съдържащи поотделно постъпващите ел.
5 – 6	5 -6	* обединение на множествата (ако са
2 – 9	2 – 3 – 4 – 9	различни) , съдържащи 2-та ел.
5 – 9	5 – 9	
7 – 3	7 – 3	
4 – 8	4 – 8	
5 – 6	5 – 6	
0 – 2	0 – 8 – 4 – 3 – 2	
6 – 1	6 – 1	

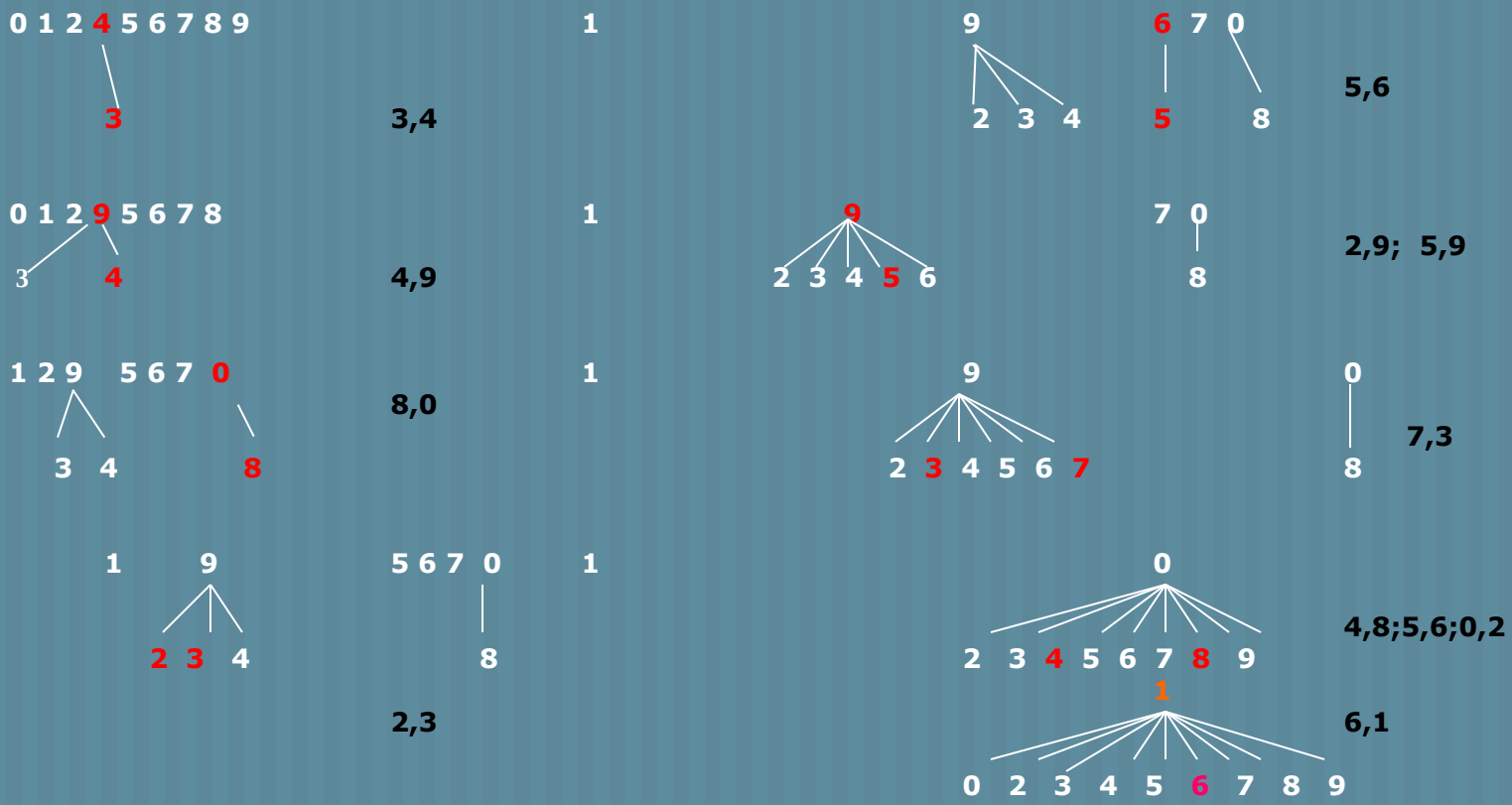
1. **Прост (начален) алгоритъм за вярно решение:**
   запомняне на всички двойки ?!(много, ресурсоемен)
2. **Алгоритъм за бързо намиране (бавно обединение)**

P	q	0 1 2 3 4 5 6 7 8 9
3	4	0 1 2 **4** 4 5 6 7 8 9
4	9	0 1 2 **9 9** 5 6 7 8 9
8	0	0 1 2 9 9 5 6 7 **0** 9
2	3	0 1 **9** 9 9 5 6 7 0 9
5	6	0 1 9 9 9 **6** 6 7 0 9
2	9	0 1 9 9 9 6 6 7 0 9
5	9	0 1 9 9 9 **9 9** 7 0 9
7	3	0 1 9 9 9 9 9 **9** 0 9
4	8	0 1 **0 0 0 0 0 0** 0 **0**
5	6	0 1 0 0 0 0 0 0 0 0
0	2	0 1 0 0 0 0 0 0 0 0
6	1	**1 1 1 1 1 1 1 1 1 1**

```
#include <stdio.h>
#define N 10 000
main ()
{           int I, p, q , t, id[N];
            for( I = 0; I < N; I++) id[I] = I;
        while(scanf("%d, %d \n", &p,&q) == край
             {
            if( id[p] == id[ q ] )continue;
            for( t = id[p], I = 0; I<N; I++)
            if(id[I] == t) id[I] = id[q];
            printf("нова връзка") }}
```

за N обекта и M обединения (вх. двойки) → поне MN инструкции (for с по 1 оп.).

Дървовидно представяне на алгоритъма:



*връзката е това което се помни. свързаност

* ако се появи 1 – 7 лесно се проверява за

Алгоритъм за бързо обединение (по-бавно намиране)

- в id[I] – указател към свързан или към себе си
- няма цикли
- алгоритъмът работи с указатели → бързина
- 2 обекта са свързани, ако указателите ги водят към общ обект (сочещ себе си)
- обединението е бързо → променя се 1 указател.

алгоритъм:

за всяко p и q
1. следваме указатели от p докато id[I] == I
2. следваме указатели от q докато id[j] == j
3. ako I != j → id[I] = id[j]

следователно обединението е просто прилепяне на дърво към дърво

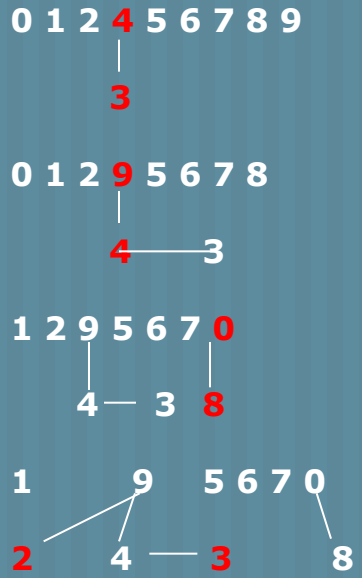
за всяко p и q:

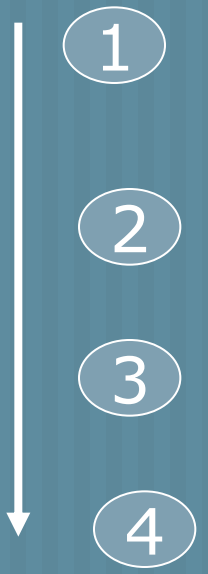
```
for ( I = p;  I != id[I];  I = id[ I ]);
for ( j = q; j != id[ j ];  j = id [ j ]);
if( I == j )  continue;              // двата указат. са еднакви  → свързани числа
id[I ] = j;                          // обединяваме
printf( " %d  %d \n", p, q);
```

p	q	0 1 2 3 4 5 6 7 8 9
3	4	0 1 2 **4** 4 5 6 7 8 9
4	9	0 1 2 4 **9** 5 6 7 8 9
8	0	0 1 2 4 9 5 6 7 **0** 9
2	3	0 1 **9** 4 9 5 6 7 0 9
5	6	0 1 9 4 9 **6** 6 7 0 9
2	9	0 1 9 4 9 6 6 7 0 9
5	9	0 1 9 4 9 6**9** 7 0 9
7	3	0 1 9 4 9 6 9 **9** 0 9
4	8	0 1 9 4 9 6 9 9 0 **0**
5	6	0 1 9 4 9 6 9 9 0 0
0	2	0 1 9 4 9 6 9 9 0 0
6	1	**1** 1 9 4 9 6 9 9 0 0

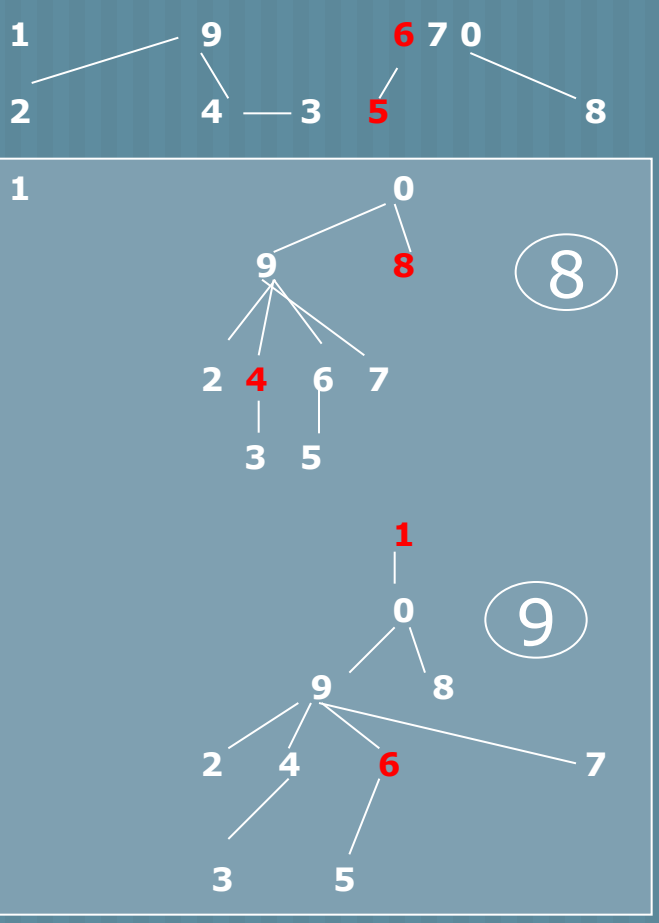
ако постъпят:

5	8	1 1 9 4 9 6 9 9 0 0





Следователно 5 и 8 са свързана двойка


5
1 9 6 7 0
2 4 3 5 8
6
1 9 7 0
2 4 6 8
3 5
8
1 0
9 8
2 4 6 7
3 5
7
1 9 0
2 4 6 7 8
3 5
9
1
0
9 8
2 4 6 7
3 5

Дискусия:

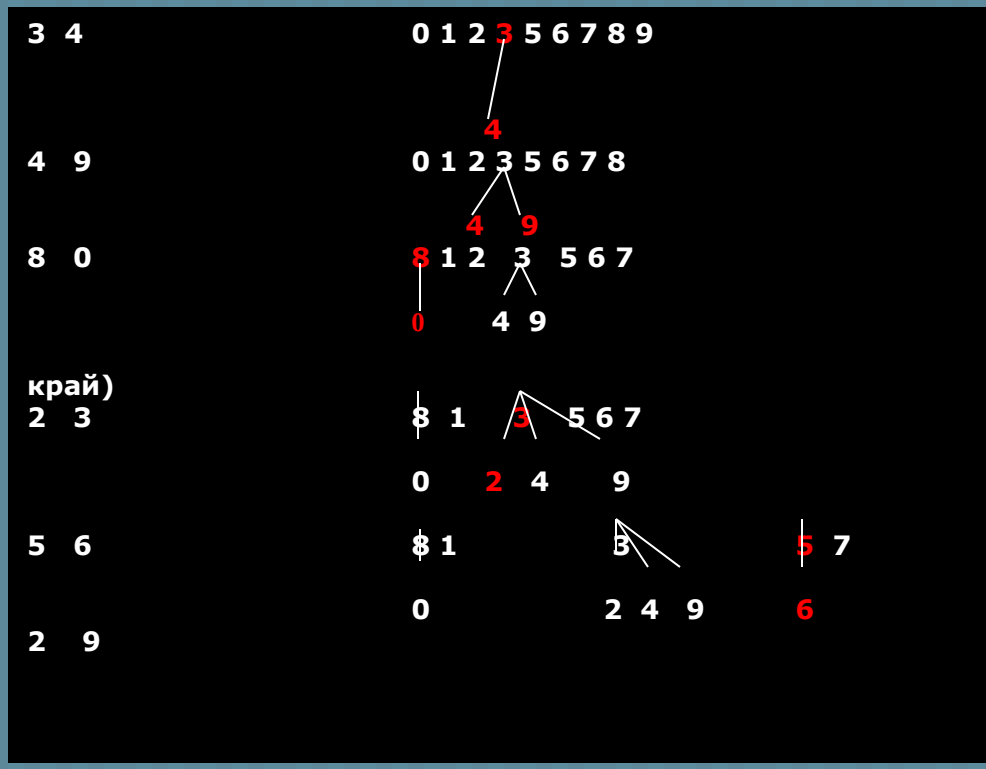
- свързани ел. → част от дървета
- 2 обекта в дърво са свързани тогава и само тогава, ако са свързани във входа
- всяко дърво има точно 1 корен
- за всеки възел на дървото има път до корена

В алгоритъм 1 достигаме корена през 1 връзка; при алгоритъм 2 --- не задължителн

бързодействие при алгоритъм 2
за всяка вх. двойка не се обхожда целия масив , както в 1.
При M двойки и N обекта и M > N има MN / 2 инструкции.
Доказателство: при входна поредица: 1 -- 2 ; 2 – 3; .... (тежка) получаваме дърво
права линия от N свързани обекта.

→ има N - 1 указателя
→за първите N обекта средно обхождаме N - 1 / 2 указателя;
→за M двойки: $MN / 2$

3. Претеглено бързо обединение
(корен на по-малко дърво към корен на по-голямо)




```
#include  <stdio.h>
#define  N   10 000
main()                                  указатели
{                                       брой в/хове
  int I, j, p, q, id[N],  sz[N];
for(I=0;I<N;  I++)
{id[I] = I;    sz[ I] = 1;}
while(scanf("%d %d\n", &p, &q)  ==

{
for( I = p; I != id[ I ]; I = id[ I ]);
for( j = q;  j != id[j ];   j = id[ j ]);

  if (I == j ) continue;
  if( sz [ I ]  < sz [ j ])
       { id[ I ] = j;  sz[ j ] += sz[ I ];}

else   { id[ j ] = I;  sz [ I ] += sz[ j ];}
  printf ( % d  %d \n", p, q);
}}
```

5 9

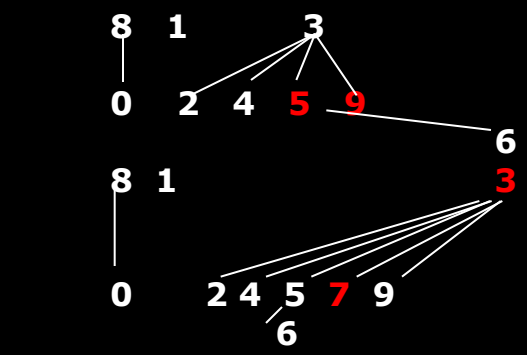
7 3

4 8

5 6

0 2

6 1



* пътищата силно се скъсяват
* най-тежък случай: еднакви дървета за обединение

0 2 4 6 8 по 2

1 3 5 7 9

0 4 8 по 4

1 2 5 6 9

3 7

по 8

(по степен на 2)

максимален брой указатели до корен при 2 ^n върха е n.

2 дървета s 2^n върха → 2^(n+1) в/

→ разстоянието нараства само на

n + 1 .

свойство: алгоритъмът проследява най-много 2 lgN указателя за да определи дали 2 измежду N обекта са свързани.

доказателство: за обект измежду **k обекта →max lg k указателя. След обединение с друго дърво**
(i + j върха, i< =j) в най-тежкия **случай - 1+lgi указателя.** Но сега за i + j елемента
Това твърдение се запазва винаги, защото:

$$1+ \lg i <= \lg(i+i) <= \lg(i +j)$$

Следователно за **всички M двойки →** **max M lg N инстукции.**

Много по-добро от преди, когато беше: **(M*N) / 2.**
За големи числа --- сега почти линеен алгоритъм.

възможни подобрения:
***сплескване на дървото като всеки връх сочи директно корена (чрез смени на указатели)**
*** Постепенно сплескване на дървото след всяко обхождане**
*** всяко ребро се разглежда 1 път (добро за on-line алгоритми) ---- не е нужно да се помни друго освен обект → пести се памет.**