Problema da conectividade

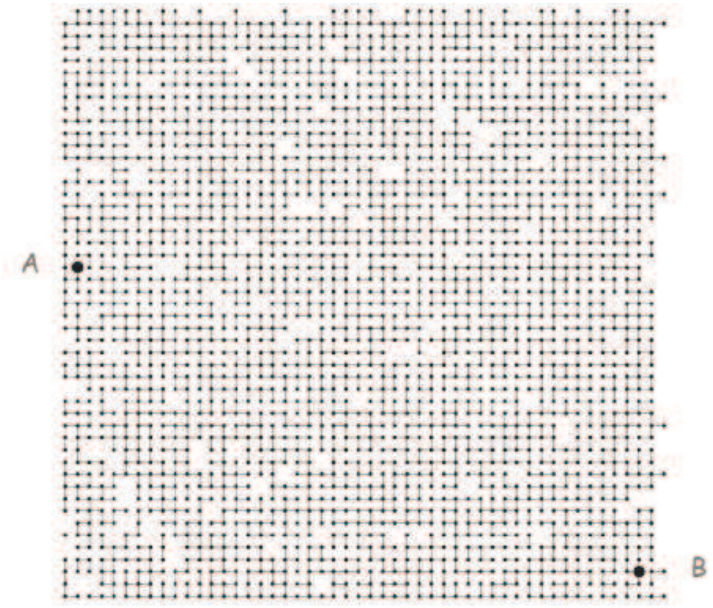
Algoritmos e Estuturas de Dados

Inverno 2006

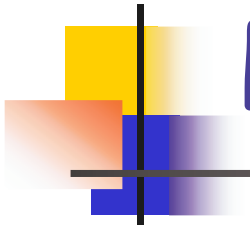
Cátia Vaz

Problema

- Nós em cada ponto de uma grelha.
- Conexões entre pares de nós.
- Existe um caminho de A para B?



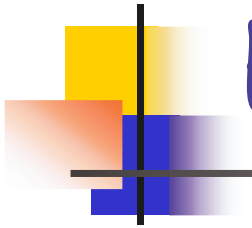
Cátia Vaz



Problema

- Instância:
  - uma sequência de N números inteiros 0, 1, …, N-1 que representam os nós;
  - um conjunto de pares (p,q) tais que $0 \leq p, q \leq N-1$ e que representam as conexões;
  - dois números inteiros A e B.

- Questão:
  - existe um caminho entre A e B, i.e., existe uma sequência de pares (p,q) da forma (A,x1)(x1,x2)…(xk,B)?

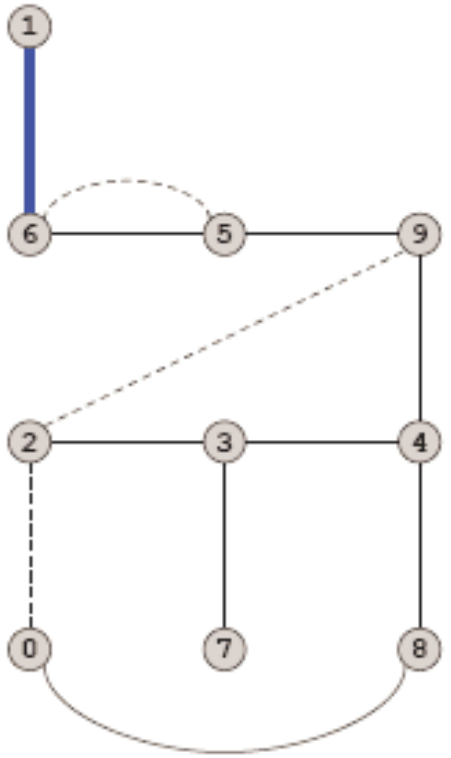


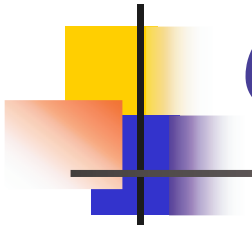
Exemplos de aplicação

- Os nós podem ser computadores numa rede e os pares podem representar conexões na rede.
  - Neste caso o algoritmo pode ser utilizado para determinar se existe uma ligação entre dois pontos na rede.
- Os nós podem ser pontos de contacto numa rede eléctrica e os pares representarem fios de conexão entre os pontos.
  - Neste caso o algoritmo pode ser utilizado para aferir a necessidade de novas ligações para assegurar a conectividade pretendida no circuito.

Exemplo

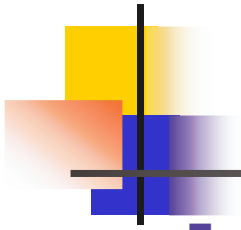
in	out	evidence
3 4	3 4	
4 9	4 9	
8 0	8 0	
2 3	2 3	
5 6	5 6	
2 9		(2-3-4-9)
5 9	5 9	
7 3	7 3	
4 8	4 8	
5 6		(5-6)
0 2		(2-3-4-8-0)
6 1	6 1	





Operações

- **Objectos:** 0 1 2 3 4 5 6 7 8 9
- **Conjuntos disjuntos:** 0 1 2-3-9 5-6 7 4-8
- **Procura:** os objectos 2 e 9 estão no mesmo conjunto? Sim, 2-3-9.
- **União:** união de conjuntos disjuntos, e.g., a união do conjunto do 3 com o conjunto do 8 é 2-3-4-8-9.

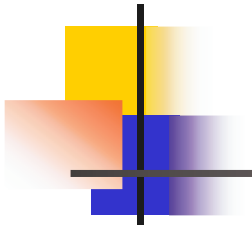


Objectivo

- Obter estruturas de dados eficientes para implementar as operações procura e união.
- Note-se que o número de objectos, N, pode ser muito grande.
- Por outro lado, o número de conexões, M, também pode ser muito grande.
- Utilizando inteiros para representar os objectos, podemos aceder facilmente a posições em arrays!

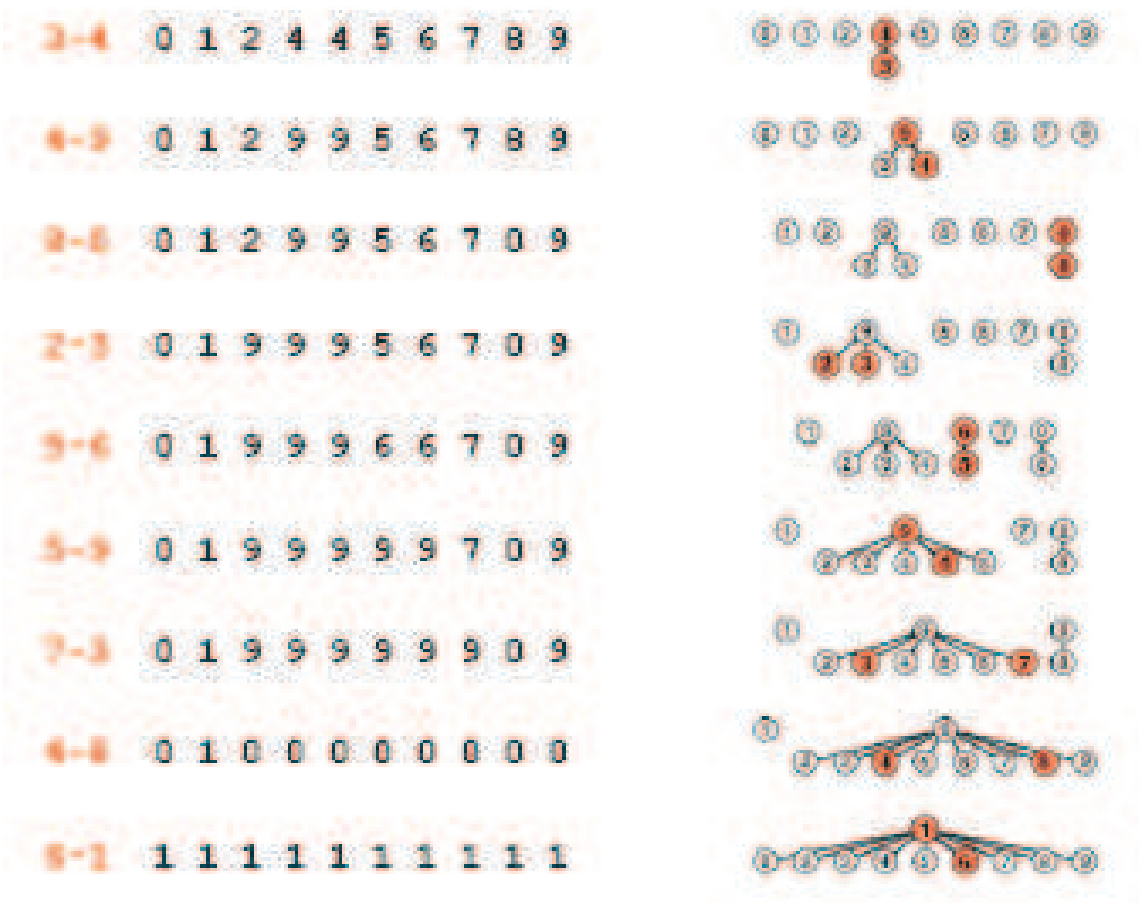
Procura Rápida *(QuickFind)*

Estrutura de dados



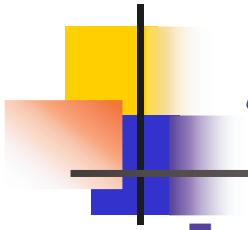
- Array de inteiros id[] de tamanho N.
- Interpretação:
  - os objectos p e q estão ligados se id[p]=id[q];
  - **operação procurar:** basta verificar se os pontos p e q tem o mesmo id;
  - **operação união:** para juntar o conjunto que contém o p com o conjunto que contém o q basta alterar o id dos elementos em ambos os conjuntos por forma a ficarem iguais (ex: id[p]=id[q]).
- Inicialização: id[p]=p

Exemplo


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

Implementação em Java

```
public class ProcuraRapida {
   private int id[];

   public ProcuraRapida(int N){
      id = new int[N];
      for(int i=0; i<N; i++) id[i]=i;
   }

   public boolean procura(int p, int q){
      return id[p]==id[q];
   }

   public void uniao(int p, int q){
      int pid = id[p];
      for(int i=0; i < id.length; i++)
         if(id[i]==pid) id[i]=id[q];
   }
}
```

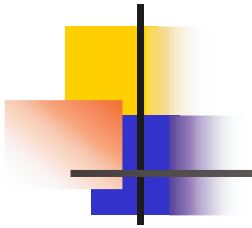


Análise

- Problema grande: 10^9 nós com 10^10 conexões.
- ProcuraRapida pode executar cerca de 10^20 operações, admitindo que resolve apenas apenas 10 instâncias do problema, i.e., NxM operações..
- Portanto, demoraria cerca de 3000 anos num computador que realize 10^9 operações por segundo.
- É necessário melhorar!

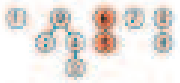
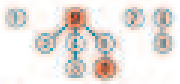
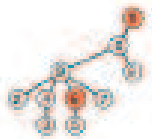
União Rápida *(UnionFind)*

Estrutura de dados



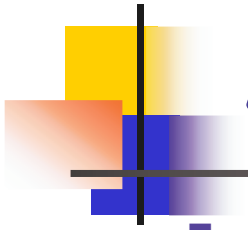
- Array de inteiros id[] de tamanho N.
- Interpretação:
  - id[p] representa o pai de p;
  - a raiz de p pode ser obtida por id[id[... id[x]...]].
  - **operação procurar:** basta verificar se p e q têm a mesma raiz;
  - **operação união:** para juntar o conjunto que contém o p com o conjunto que contém o q atribuir o id da raiz de q à raiz de p (id[raiz(p)]=id[raiz(q)]);
- Inicialização: id[p]=p

Exemplo

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	
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	
6-1	1 1 9 4 9 6 9 9 0 0	

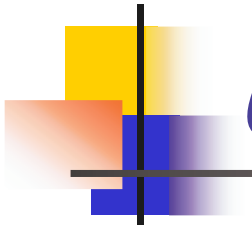
Implementação em Java

```
public class UniaoRapida {
   private int id[];
   public UniaoRapida(int N){ ... }

   private int root(int x){
      while(x!=id[x]) x = id[x];
      return x;
   }

   public boolean procura(int p, int q){
      return root[p]==root[q];
   }

   public void uniao(int p, int q){
      int i=root(p), j=root(q);
      if(i!=j) id[i]=j;
   }
}
```



Análise

- Problemas:
  - **ProcuraRapida**: união demora demasiado tempo; as árvores estão equilibradas, mas exige um grande número de operações.
  - **UniaoRapida**: procura demora demasiado tempo; as árvores não estão equilibradas.
- **ProcuraRapida**:
  - **uniao**: N operações;
  - **procura**: 1 operação:
- **UniaoRapida**:
  - **uniao**: 1 operação quando se juntam raizes;
  - **procura**: N operações.



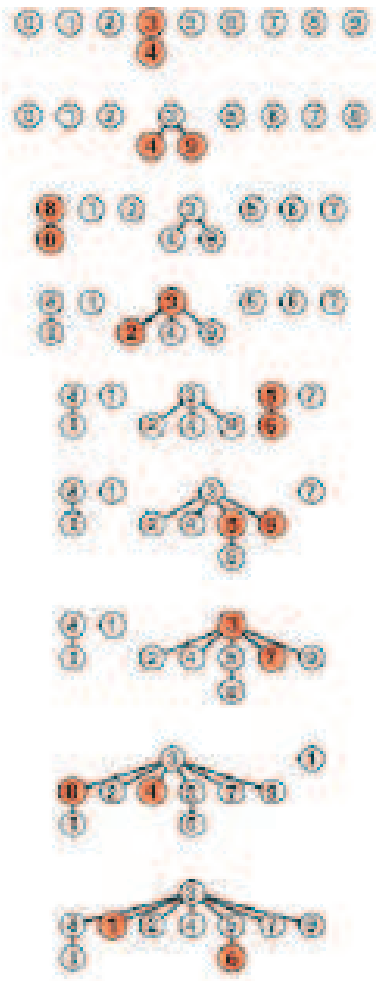
União Rapida com pesos
(Weighted Quick-Union)

- Objectivo: modificar **`UniaoRapida`** por forma a garantir árvores equilibradas.

- Solução:
  - guardar o tamanho de cada componente;
  - procurar equilibrar as árvores colocando as de menor tamanho por baixo na operação uniao.
    - sem perda de generalidade, se forem de igual tamanho, id[raiz(q)]=id[raiz(p)].

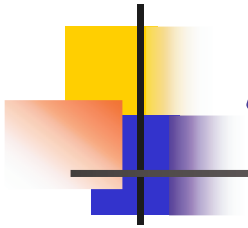
Exemplo

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



Implementação em Java

```
public class UniaoRapidaPesos {
   private int id[], tm[];

   public UniaoRapidaPesos(int N){
      id = new int[N]; tm = new int[N];
      for(int i=0; i<N; i++){ id[i]=i; tm[i]=1; }
   }
   private int root(int p){ ... }
   public boolean procura(int p, int q){ ... }

   public void uniao(int p, int q){
      int i=root(p), j=root(q);
      if(i==j) return;
      if(tm[i]<tm[j]) {id[i] = j; tm[j]+=tm[i];}
      else {id[j]=i; tm[i]+=tm[j];}
   }
}
```

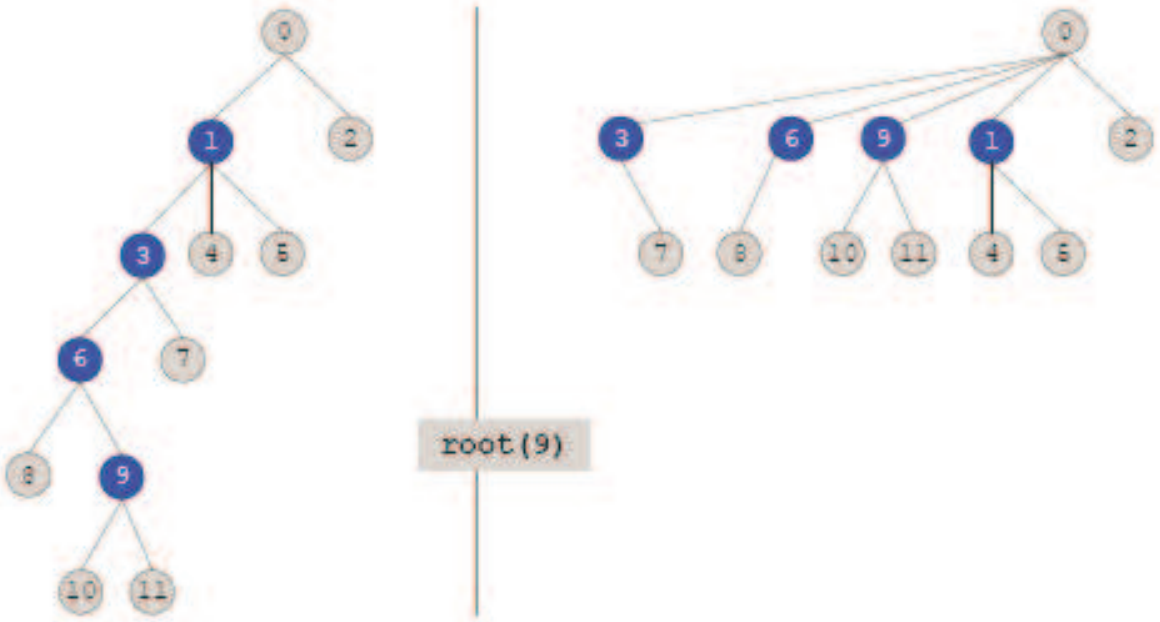


Análise

- **procura**: leva tempo proporcional à profundidade de p e q.
- **uniao**: tempo constante se p e q forem raizes, caso contrário é o caso idêntico ao da procura.
- A profundidade (nível) é no máximo log N.
- Portanto:
  - **procura**: log N operações;
  - **uniao**: log N operações.
- Pode ser melhorado! ☺

Path Compression

- Após computar a raiz de x, alterar o id de cada nó examinado para raiz de x.
  - Exemplo:



Weighted Quick-Union with Path Compression by halving

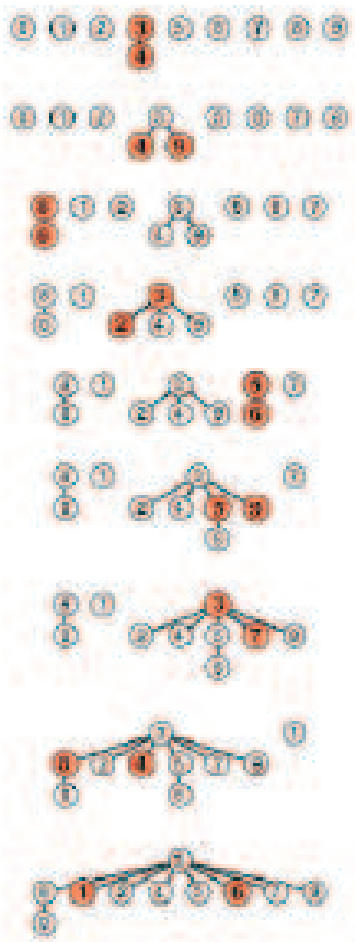
- Alteração ao **UniaoRapidaPesos:**

```
public int root(int x){
  while(x!=id[x]){
        id[x]=id[id[x]];
        x=id[x];
 }
return x;
}
```

- Exercício: Implementar com *Full Path Compression.*

Exemplo

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



Análise

- Uma sequência de M operações de união e procura em N elementos demora O(N + M lg* N) .

N	2	4	16	65536	2^{65536}
lg*N	1	2	3	4	5

- Problema grande: 10^9 nós com 10^10 conexões.
- Admitindo que resolve apenas apenas 10 instâncias do problema, i.e., NxM operações:
  - reduz o tempo de 3000 anos a 1 minuto.