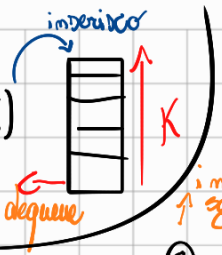
CODA

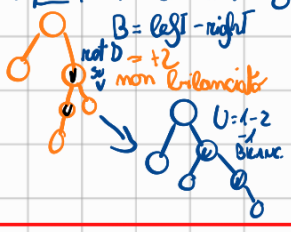


- enqueue O(K)
- dequeue O(1)

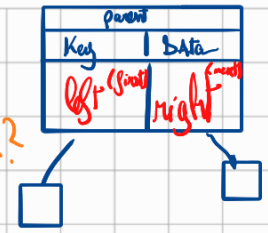
Alberi REMINDER BINARI, RICERCA o AVL

AVL: Bilanciati ± 1 / 0

B = left - right



COME FUNZIONA?



BFS(T)

```
Q = new Queue;
if T.root ≠ NIL
    q.enqueue(T.root)
While Q is not empty
    x = q.dequeue
    visit(x)
    if x.left ≠ NIL
        q.enqueue(x.left)
    if x.right ≠ NIL
        q.enqueue(x.right)
```

DFS(T)

```
DFS(T.left)
preorder / post O(n)
if T ≠ NIL
    visit(T) (se post in fondo)
    Preorder(T.left)
    Preorder(T.right)
```

UnionFind — MakeSet → crea un insieme

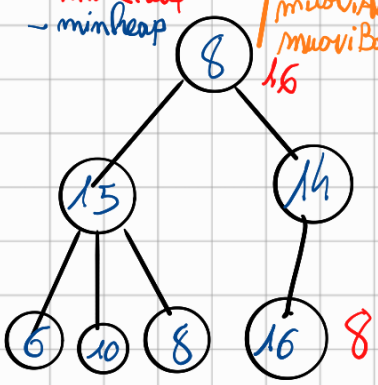
QuickFind
makeset, find → O(1)
union → O(n) | euristica peso UNION O(log n) ammort.

QuickUnion
makeset, union → O(1)
find → O(n) | euristica rango FIND O(log n)

D HEAP

- maxheap
- minheap

muoviAlto / muoviBasso



CONSIGLI PER NON SBAGLIARE

increase | decrease

⇒ I D-HEAP devono sempre avere le PROPRIETÀ DI STRUTTURA (completamento a sx) E DI ORDINE (relazione padre-figlio) RISPETTATE

→ CONTROLLARE DOPO OGNI OPERAZIONE ESEGUITA

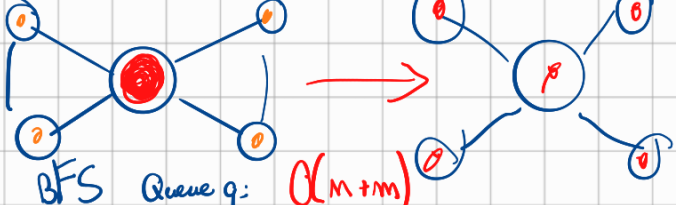
Grafi

DFS:


```
DFS := Graph(V,E)
foreach u in V
    u.marked = w;
    u.parent = NIL;
foreach u in V
    if u.marked = w
        DFS_visit(u);
```

```
DFS_visit  O(n+m)
v.marked = g;
v.dt = Time; time++;
for all u adjac to v
    if u = w
        DFS_visit(u)
visit(v)
v.ft = time++;
v.mark = black;
```

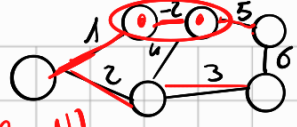
BFS



BFS Queue q: $O(n+m)$

```
foreach v in V
    v.marked = false;
Tree T = s
q.enqueue(s);
s.dist = 0
s.marked = true
while !Q.isEmpty
v = q.dequeue
for all u adjac to v
    if u.marked = false;
        u.dist = v.dist + 1
        u.marked = true;
        u.parent = v; q.enqueue(u)
return Tree;
```

GRAFI MST



: parto da qui

Kruskal $O(E \log N)$

```
UF uf
Tree T
for v in V
  uf.makeset(v)
E.sort();
for (u,v) in E
  Tu = uf.find(u)
  Tv = uf.find(v)
  if Tu ≠ Tv
    add(Tree,(u,v))
    uf.union(Tu,Tv)
return Tree;
```

CAMMINI MINIMI

- BELLERMAN E FORD NO CICLI E INPUT modo
- DIJKSTRA migliore NO PESI NEG
- FLOYD WARSHALL grafo orientato no cicli

- BELLERMAN E FORD $O(nm)$

```
n = G.numNodi()
int pred[1...n], v, u;
double D[1..N];
for v to n
  D[v] = ∞
  pred[v] = -1;
D[s] = 0;
for i to n-1
  foreach (u,v) in E
    if(D[u] + w(u,v) < D[v]
      D[v] = D[u] + w(u,v)
      pred[v] = u;
foreach (u,v) in E
  if D[u] + w(u,v) < D[v]
    return "Cicli negativi"
return D;
```

s (source)

DIJKSTRA $O(m \log n)$

```
n = G.numNodi()
int pred[1...n], v, u;
double D[1..N];
for v to n
  D[v] = ∞
  pred[v] = -1;
D[s] = 0;
CodaPriorità Q; Q.insert(s, D[s]);
while !Q.isEmpty
  u = Q.find(); Q.deleteMin();
  for v adjac u
    if(D[v] == ∞)
      D[v] = D[u] + w(u,v);
      Q.insert(v, D[v])
      pred[v] = u;
    else if (D[u] + w(u,v) < D[v])
      Q.decreaseKey(v, D[u] + w(u,v));
      D[v] = D[u] + w(u,v)
      pred[v] = u;
return D;
```

Floyd Warshall $O(n^3)$ spazio $O(n^2)$

PROG. DIN

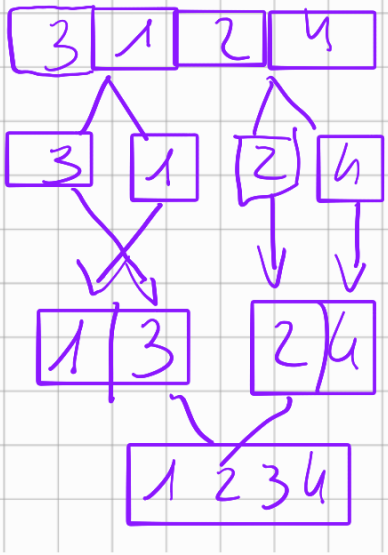
```
int n = V.nVert
double S[1...n, 1..n]
Vertex next[1...n, 1..n]
for x to n
  for y to n
    if x == y
      S[x,y] = 0
      next[x,y] = -1
    else if (x,y) ∈ E
      S[x,y] = w(x,y)        // ragg.
      next[x,y] = y
    else
      S[x,y] = ∞             // non ragg
      next[x,y] = -1;
for K to n
  for x to n
    for y to n
      if S[x,K] + S[K,y] < S[x,y]
        S[x,y] = S[x,K] + S[K,y];
        next[x,y] = next[x,K];
return S;
```

Divide Et Impera top-down

Problema su array

(mergesort) Mergesort(T[1..N] array, int i, int f)

```
if len = 1
  return array;
mid = len(array)/2;
mergesort(array, i, mid)
mergesort(array, mid+1, f)
Merge(v, i, mid, f);
```



Greedy

SCELGO OPZIONE MIGLIORE

Es: Resto o Dijkstra

```
Resto-Greedy (R, int tagli[])
mm = 0; i = 1;
while R>0 and i≤n
    if R ≥ tagli[i]
        R = R - tagli[i]
        mm++;
    else
        i++;
if R>0
    ERR(IMPOSSIBILE)
else
    return mm;
```

Trovata sol migliore

← caso resto solo canonico

SI DOVREBBE CAPIRE come scriverla molto semplice

Programmazione Dim:


```
for i to m
    for j to m
        if i==0 & j==0
            sol[0][0] = M.val[0][0]
        // CASO 1 PRIMA RIGA
        else if (j==0)
            sol[i][0] = M.val[i][j] + sol[i-1][0];
        // CASO 2 PRIMA COLONNA
        else if (i==0)
            sol[0][j] = M.val[i][j] + sol[0][j-1]
        else // CASO GENERALE
            sol[i][j] = M.val[i][j] + max(sol[i-1][j], sol[i][i])
```

ZAINO

```
for int i to n
    for j to m
        if (j==0){S[i][0]=0;   // PRIMA RIGA 0 spazio disp quindi 0
        else if (X[i]<=j)      // peso oggetto ≤ quello che posso prendere
            if (i==0) S[i][j] = X[0];
            else // SENNO CONTROLLO MAX   (prima colonna prendo)
            S[i][j] = MAX(X[i]+S[i-1][j-X[i]], S[i-1][j]);
        else
            if i==0 S[i][j]=0;
            else SENNO È UGUALE ALLA PREC
                S[i][j]=S[i-1][j];
```

Sono simili ma cambiano gli if

$O(n^2)$