

A thick black L-shaped frame is positioned on the left and right sides of the slide, framing the central text.

COMPLEMENTI DI SHORTEST-PATH

SDAC 24

Fabrizio d'Amore

problemi di shortest-path (SP)

- grafo di riferimento $G=(V,E)$
 - *semplice/orientato*
 - *pesato*
 - pesi non negativi
 - pesi arbitrari, ma no cicli a peso negativo
 - *nessuna restrizione*
 - *non pesato*
- problemi SP
 - $SP(G, u, v)$ (*shortest-path in G , da u a v*)
 - $SSSP(G, s)$ (*determinare shortest-path tree radicato in s*)
 - $APSP(G)$ (*determinare uno shortest-path per ogni coppia $(u,v) \in V^2$*)

grafi non pesati

- algoritmo può essere basato su BFS
- non è necessario costruire le liste L_i dei nodi a distanza i dalla sorgente (SSSP)
- $\text{BFS}(G,s)$ costruisce SP-tree radicato in s , descritto dai pred
- nel caso di SP semplice (da u a v) si può eseguire una variante $\text{BFS}(G, u, v)$ che continua fino all'inserimento di v nella coda, quindi termina
- utilizzabile su grafi semplici e orientati
 - *incidentEdges* deve restituire gli archi uscenti, nel caso di grafi orientati

```
BFS(Graph G)
  forall v ∈ V(G) v.label = unexplored
  forall e ∈ E(G) e.label = unexplored
  forall v ∈ V(G)
    if (v.label == unexplored) BFS(G, v)
```

```
BFS(Graph G, Vertex v)
  Q = new empty queue()
  v.pred = 0
  v.label = visited
  Q.insert(v)
  while (!Q.isEmpty())
    w = Q.dequeue()
    forall e ∈ w.incidentEdges()
      if (e.label == unexplored)
        x = e.opposite(w)
        if (x.label == unexplored)
          x.pred = w
          e.label = tree
          x.label = visited
          Q.enqueue(x)
        else e.label = cross
```

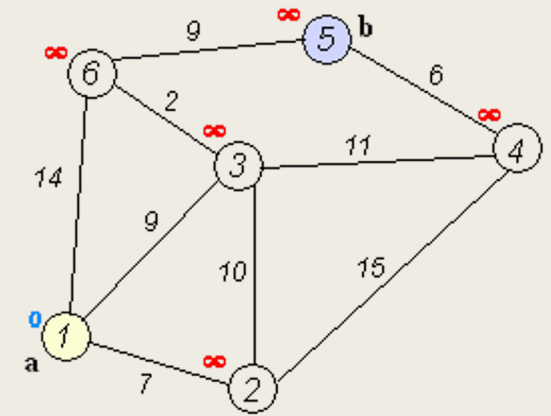
estrazione path da etichette

- terminata la visita, lo SP da u a v può essere così restituito

```
path = new empty list()
p = v
while(p != 0)
    path.addFirst(p)
    p = p.pred
return path
```

SP e SSSP su grafi pesati (pesi non negativi)

- Dijkstra sia per grafi semplici che orientati
- basato su rilassamento di archi
- prev descrive SP-tree
- nel caso di SP da source a dest si può usare la variante che termina quando si estrae dest dallo heap
- la ricostruzione dello SP può avvenire come nella slide precedente



```
Dijkstra(Graph G, Vertes source)
dist[source]=0
prev[source]=0
H=new empty min-heap()
forall v∈V(G)
    if(v≠source)
        dist[v]=INFINITY
        prev[v]=UNDEFINED
        H.add(v, dist[v])
while(!H.isEmpty())
    u=H.extractMin()
    forall neighbor v of u (s.t. v∈H)
        alt=dist[u]+weight(u, v)
        if(alt<dist[v])
            dist[v]=alt
            prev[v]=u
            H.decreasePriority(v, alt)
return dist[], prev[]
```

SSSP con pesi arbitrari (no cicli a costo < 0)

- Algoritmo di Bellman-Ford
- Anch'esso basato su rilassamento di archi, ma non usa heap
 - *tenta tutti i possibili rilassamenti $n-1$ volte (perché lo SP con il maggior numero di archi ne avrà al più $n-1$)*
 - *a iterazione i del for, algoritmo trova SP di al più i archi*
 - *ciclo finale: per verificare se esiste SP di lunghezza n , il che denuncia un ciclo di peso negativo*
- Costo: $\Theta(mn)$
 - *ciclo for su tutti i nodi che contiene ciclo for su tutti gli archi*
- Se in una iterazione non si rilassano archi l'algoritmo può terminare
- Se ne usa una versione distribuita per implementare protocolli di routing (RIP)

```
BellmanFord(Graph G, Vertex source)
  forall v ∈ V(G)
    dist[v] = infinite
    pred[v] = null
  dist[source] = 0
  for i from 1 to |V(G)| - 1
    forall (u, v) ∈ E(G)
      w = weight(u, v)
      if (dist[u] + w < dist[v])
        dist[v] = dist[u] + w
        pred[v] = u
  forall (u, v) ∈ E(G)
    w = weight(u, v)
    if (dist[u] + w < dist[v])
      error "Graph contains negative-weight
cycle"
  return dist[], pred[]
```

APSP pesi arbitrari (no cicli < 0)

- Algoritmo di Floyd-Warshall, simile a quello per la chiusura transitiva
- Valido per grafi semplici e orientati
- Basato su programmazione dinamica
- Alla fine dell'algoritmo
 - *La matrice dist contiene le distanze fra le coppie di vertici*
 - *La matrice next contiene nella cella $[i][j]$ l'indice del nodo successore di i , nello SP da i a j*
 - *L'algoritmo Path ricostruisce dalla matrice next lo SP da u a v*
- Costo (evidente): $\Theta(n^3)$

```
FloydWarshall(Graph G)
  dist = new n×n matrix of distances
  // (all infinite)
  next = new n×n matrix of vertex indices
  // (all zeros)
  forall (u,v)∈E(G)
    dist[u][v]=w(u,v)
    next[u][v]=v
  for k from 1 to n
    for i from 1 to n
      for j from 1 to n
        if(dist[i][j]>dist[i][k]+dist[k][j])
          dist[i][j]=dist[i][k]+dist[k][j]
          next[i][j]=next[i][k]
```

```
Path(Vertex u, Vertex v)
  if(next[u][v]==0) return []
  path=[u]
  while(u!=v)
    u=next[u][v]
    path.append(u)
  return path
```