Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni in modalità mista o a distanza

# Algoritmo di Dijkstra

**Giancarlo Bongiovanni**
**Ivano Salvo**


SAPIENZA
UNIVERSITÀ DI ROMA

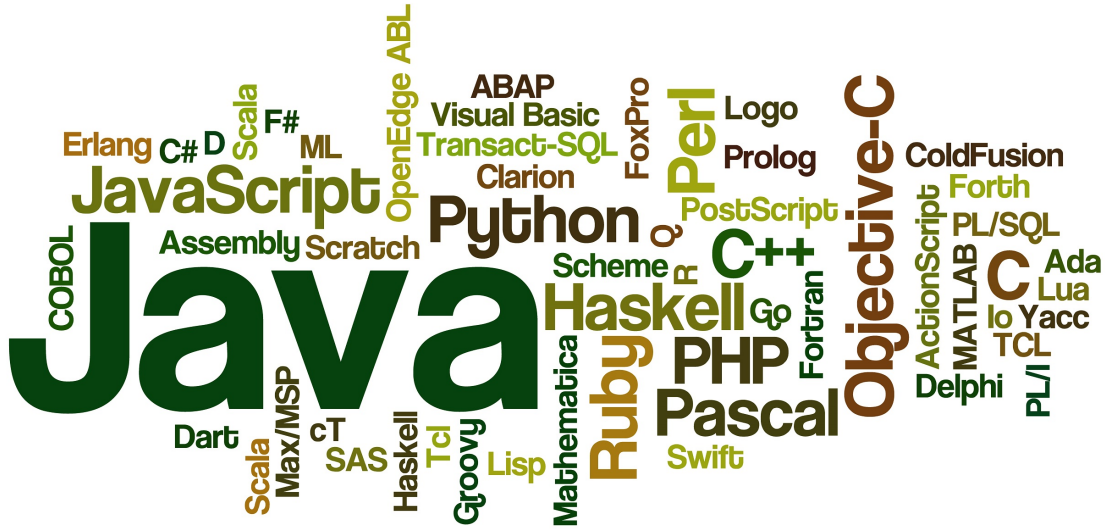
Queste dispense sono state realizzate sulla base delle slide
preparate da T. Calamoneri e G. Bongiovanni
per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

# *E adesso la pubblicità*

# *Per chi si fosse divertito…*

# *…o volesse saperne di più*

- *Linguaggi di Programmazione*, abilità informatica con sola idoneità, 3zo anno, **Matematica** (**C++, ~~Haskell~~**) **Python?**
  `https://twiki.di.uniroma1.it/twiki/view/LDPMat/WebHome`
- *Tecniche di Programmazione Imperativa e Funzionale*, 6 crediti, 3zo anno di **Informatica** (**C, Haskell**)
  `https://twiki.di.uniroma1.it/twiki/view/TPFI/WebHome`



# Reti e grafi (1)

Numerose reti di importanza vitale per la nostra società sono modellate per mezzo di grafi. Si pensi ad esempio a:

- reti autostradali;
- reti ferroviarie;
- reti per la distribuzione dell'energia elettrica;
- reti telefoniche;
- la rete Internet.

# Reti e grafi (2)

Nell'ambito di tali strutture un problema di grande rilevanza è determinare il ***cammino di costo minimo*** fra un punto ed un altro nella rete (ossia fra un vertice ed un altro nel grafo che modella la rete).

In tali scenari ogni arco del grafo ha associato un ***costo di attraversamento***, di norma strettamente positivo, la cui determinazione dipende da una combinazione delle grandezze in gioco nello specifico contesto. Ad esempio:

- distanza chilometrica e costi di trasporto e pedaggio nelle reti stradali e ferroviarie;
- ampiezza di banda trasmissiva e costi di affitto delle linee nelle reti telefoniche e di elaboratori;
- ecc.

# Reti e grafi (3)

Il problema di calcolare il cammino minimo fra un punto ed un altro nella rete è di enorme importanza nella rete Internet, perché:

- il grafo che la modella è costituito di un numero molto elevato di vertici ed archi (centinaia di migliaia);
- le condizioni operative dei collegamenti cambiano molto frequentemente, quindi anche i cammini minimi devono essere ricalcolati molto spesso (ogni pochi secondi).

Per poter approfondire almeno un poco questo discorso è necessario illustrare a grandi linee le caratteristiche principali della rete Internet.

# Algoritmo di Dijkstra (2)

L'algoritmo di Dijkstra visita i nodi nel grafo, in maniera simile a una ricerca in ampiezza o in profondità.

In ogni istante, l'insieme dei vertici del grafo è diviso in tre parti:

- ***l'insieme VIS*** dei nodi visitati, per i quali è ormai fissato definitivamente il cammino minimo;
- ***l'insieme F*** dei nodi di frontiera, che sono successori dei nodi visitati, per i quali ancora non è definitivo il cammino minimo;
- ***i nodi sconosciuti***, che sono ancora da esaminare.

Per ogni nodo v, l'algoritmo tiene traccia:

- di un valore ***dist(v)***, che rappresenta il costo noto finora del cammino minimo da s al vertice v;
- dell'indice ***pred(v)***, il cui ruolo è identico a quanto discusso nelle visite in ampiezza e profondità: esso identifica il predecessore di v nell'attuale cammino minimo da s a v.

# Algoritmo di Dijkstra (3)

L'algoritmo consiste nel ripetere la seguente iterazione finché l'insieme F non si svuota:

- si prende dall'insieme F un qualunque nodo v ***avente distanza dist(v) minima***;
- si sposta v da F in VIS;
- si inseriscono tutti i successori di v ancora sconosciuti in F;
- per ogni successore w di v si aggiornano i valori dist(w) e pred(w). L'aggiornamento viene effettuato con la regola:

***dist(w) = minimo (dist(w),(dist(v)+peso dell'arco(v,w))***

- se il valore di dist(w) è stato effettivamente modificato, allora pred(w) viene posto uguale a v, il che ci permette di ricordare che, al momento, il cammino di peso minimo che conosciamo per arrivare da s a w ha come penultimo nodo v.

# Algoritmo di Dijkstra (4)

L'algoritmo segue un'idea piuttosto naturale: se sappiamo che con peso pari a dist(v) possiamo arrivare fino a v, allora arrivare a w non può costare più di arrivare a v e spostarsi da v lungo l'arco (v,w) fino a w.

Si noti che una fondamentale differenza con la visita in ampiezza è legata al fatto che *dobbiamo mantenere i nodi in F ordinati rispetto alla loro distanza nota da s*, il che implica che la struttura d'appoggio usata nella visita non può più essere una coda ma diviene una *coda con priorità*, nella quale l'elemento da estrarre è quello avente distanza minima dal vertice s di partenza.

L'algoritmo funziona anche su grafi non orientati: è sufficiente sostituire, ad ogni arco non orientato del grafo, una coppia di archi orientati in senso opposto l'uno all'altro ed aventi ciascuno lo stesso peso dell'arco non orientato che sostituiscono.

# Algoritmo di Dijkstra – pseudocodice (1)

Lo pseudocodice utilizza alcune informazioni aggiuntive, necessarie al suo corretto funzionamento:

- un vettore ***dist[v]*** per gestire il costo del cammino minimo da s a v (inizialmente posto a ∞);
- un vettore ***pred[v]*** per memorizzare il predecessore di ciascun vertice (inizialmente posto a NULL);
- un vettore ***vis[v]*** per indicare se il vertice è o no in VIS (inizialmente posto a false);
- una ***coda con priorità q*** per gestire l'ordine di visita dei vertici: i vertici presenti via via nella coda q sono quelli dell'insieme F.

# Algoritmo di Dijkstra – pseudocodice (2)

```
Funzione Dijkstra (G: grafo pesato con pesi non negativi;
                   Q: coda con priorità)
   Assegna a tutti i nodi v
      dist [v] ← ∞; pred[v] ← NULL; vis[v] ← FALSE
   scegli il nodo di partenza s
   dist[s] ← 0
   Enqueue(Q, s)
   finché la coda Q non è vuota
      v ← Dequeue(Q) // v è il vertice su cui lavorare
      vis[v] ← TRUE  // per v il cammino minimo è definitivo
      per ogni vertice w adiacente a v tale che vis[w]=FALSE
         if dist[w] > dist[v] + peso_arco(v,w) then
            dist[w] ← dist[v] + peso_arco(v,w)
            pred[w] ← v
            if (w non è nella coda q)
               Enqueue(Q, w)
            else aggiusta la posizione di w in Q
return
```

# Algoritmo di Dijkstra – esempio (1)

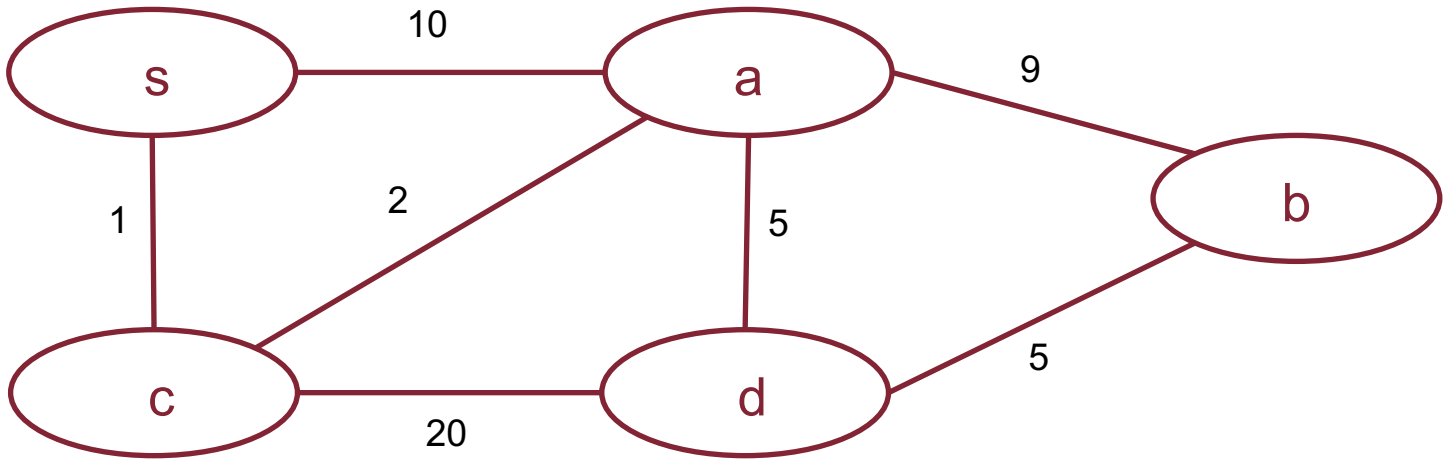
Nell'esempio si adotta la seguente simbologia:

- il ***vertice s*** è il vertice di partenza;
- i ***vertici verdi*** sono quelli in VIS;
- i ***vertici gialli*** sono quelli in F (quindi sono i vertici nella coda);
- i ***vertici arancio*** sono vertici già presenti in F i cui valori vengono aggiornati durante un'iterazione;
- all'interno di ogni vertice v si indicano i valori di dist(v) e pred (v);
- gli ***archi rossi*** sono quelli dell'albero dei cammini minimi.

# Algoritmo di Dijkstra – esempio (2)

## Situazione iniziale

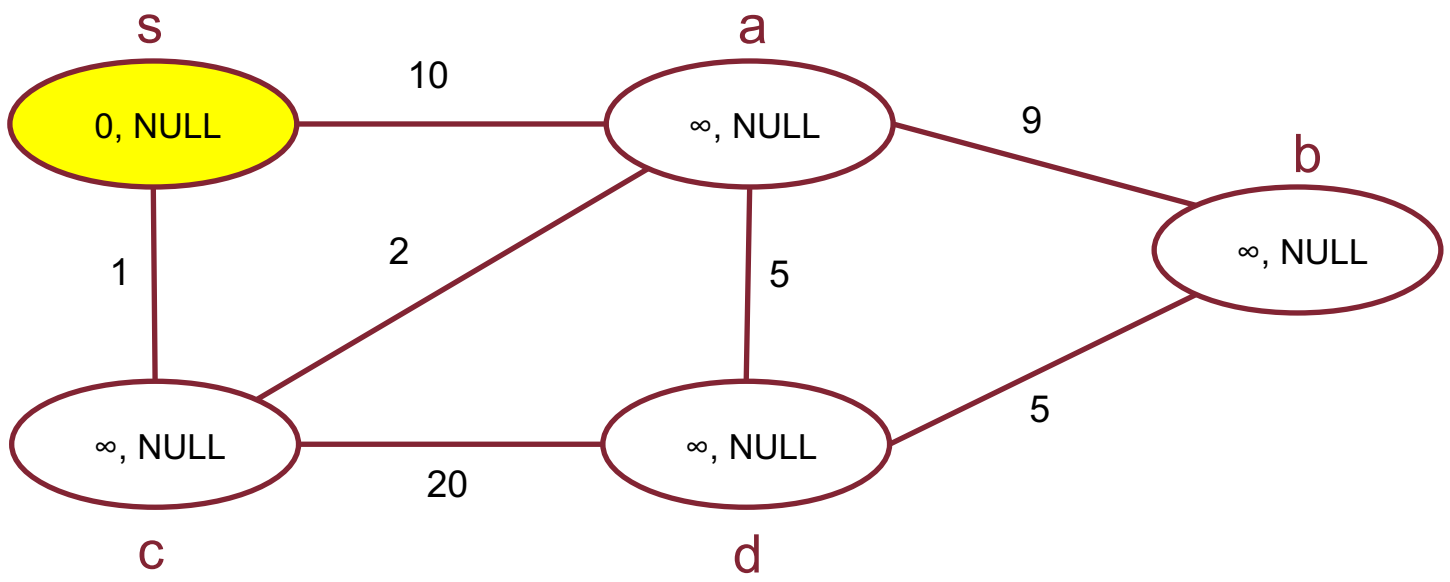
Il grafo pesato su cui eseguire l'algoritmo è il seguente:



# Algoritmo di Dijkstra – esempio (3)

## Inizializzazione

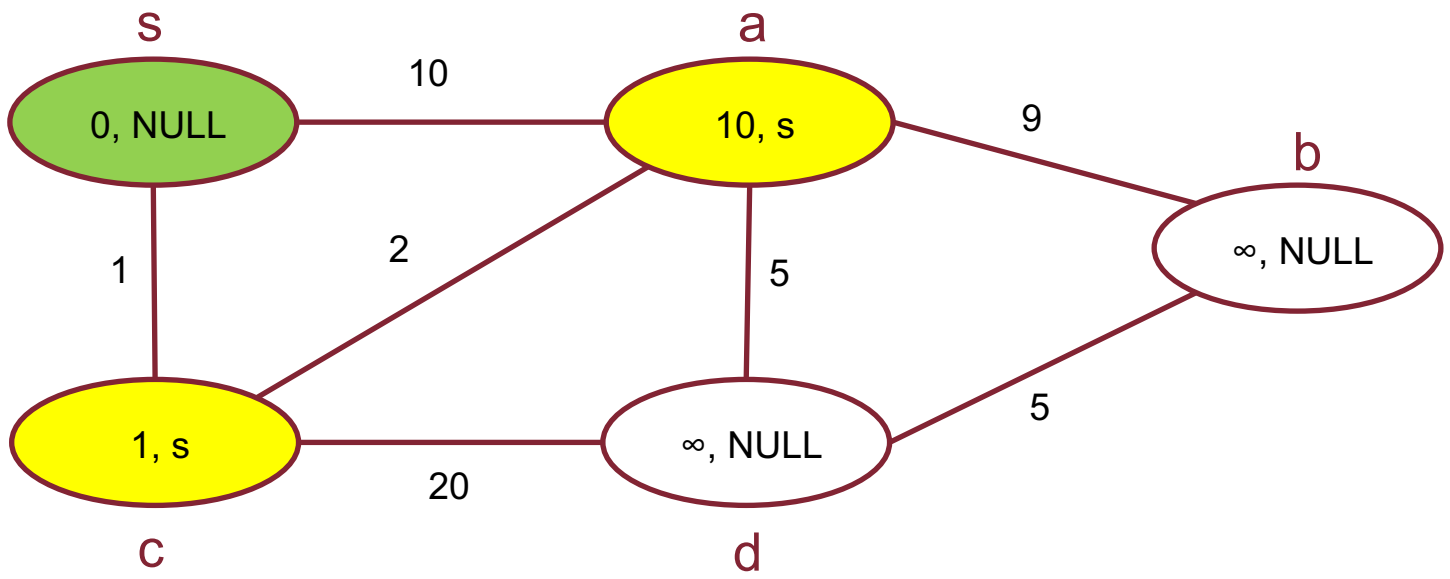
Dopo l'inizializzazione la situazione è questa (s è l'unico vertice nella coda):




# Algoritmo di Dijkstra – esempio (4)

## Prima iterazione

Nella prima iterazione del while si estrae dalla coda la sorgente (che quindi viene spostata in VIS) e si mettono in coda (e quindi in F) i suoi adiacenti, aggiornandone i valori:

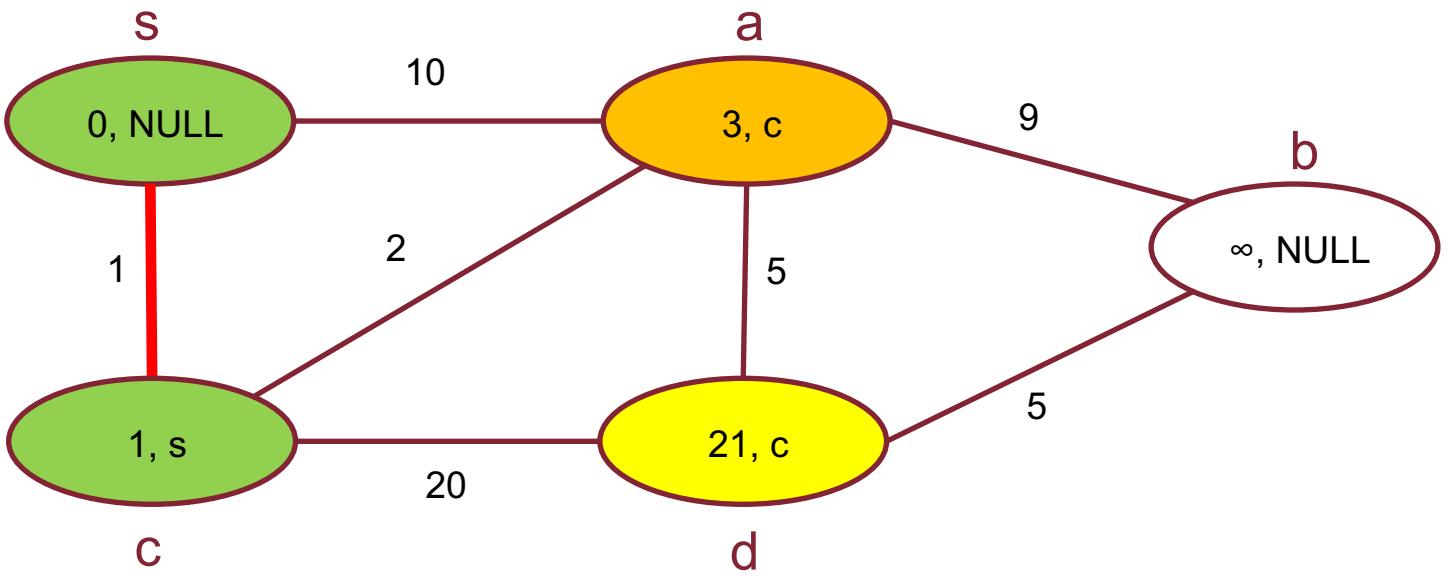




# Algoritmo di Dijkstra – esempio (5)

## Seconda iterazione

Nella seconda iterazione fra i vertici in F si sceglie quello a distanza minima (c), che viene spostato in VIS, si aggiungono i nuovi vertici (d) scoperti tramite c e si aggiornano i valori di quelli (a) già in F:

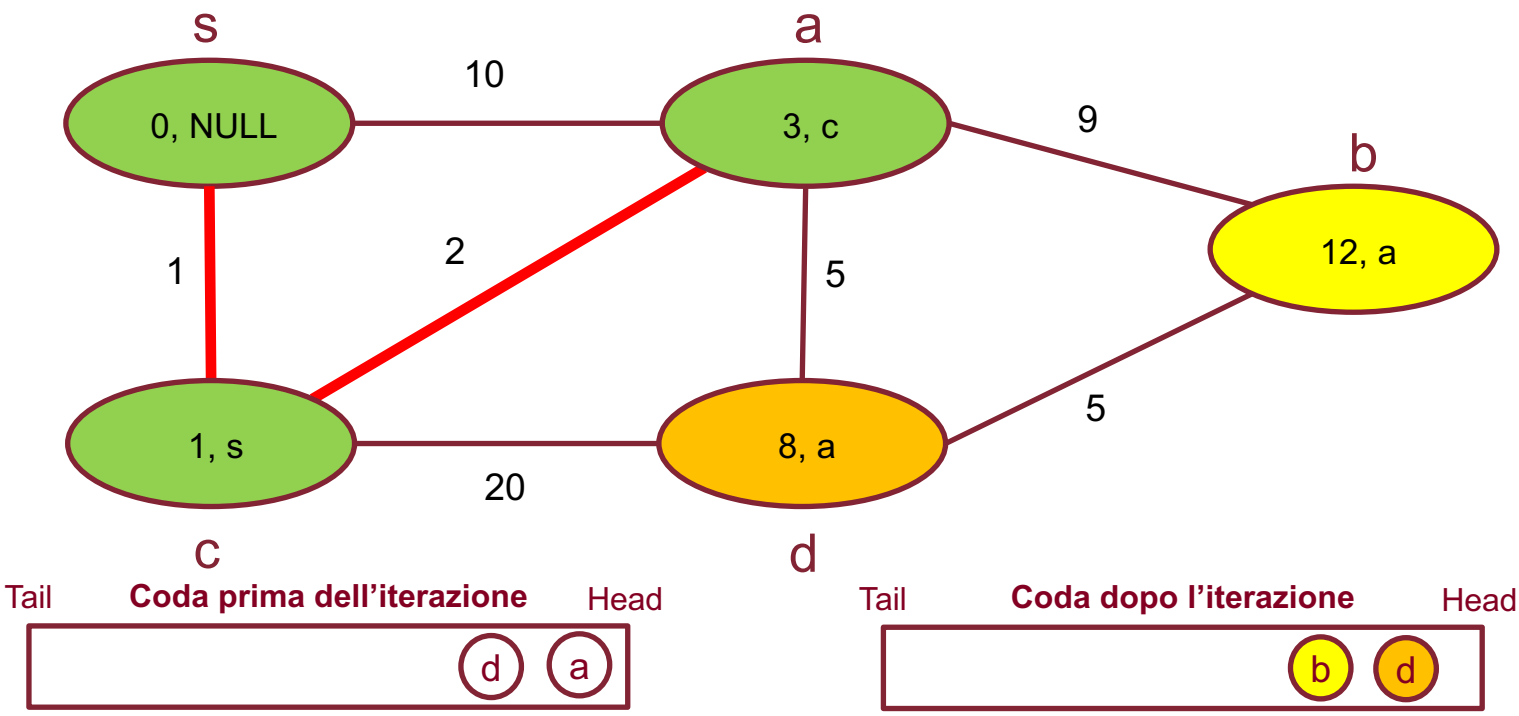




# Algoritmo di Dijkstra – esempio (6)

## Terza iterazione

Nella terza iterazione fra i vertici in F si sceglie quello a distanza minima (a), che viene spostato in VIS, si aggiungono i nuovi vertici (b) scoperti tramite a e si aggiornano i valori di quelli (d) già in F:

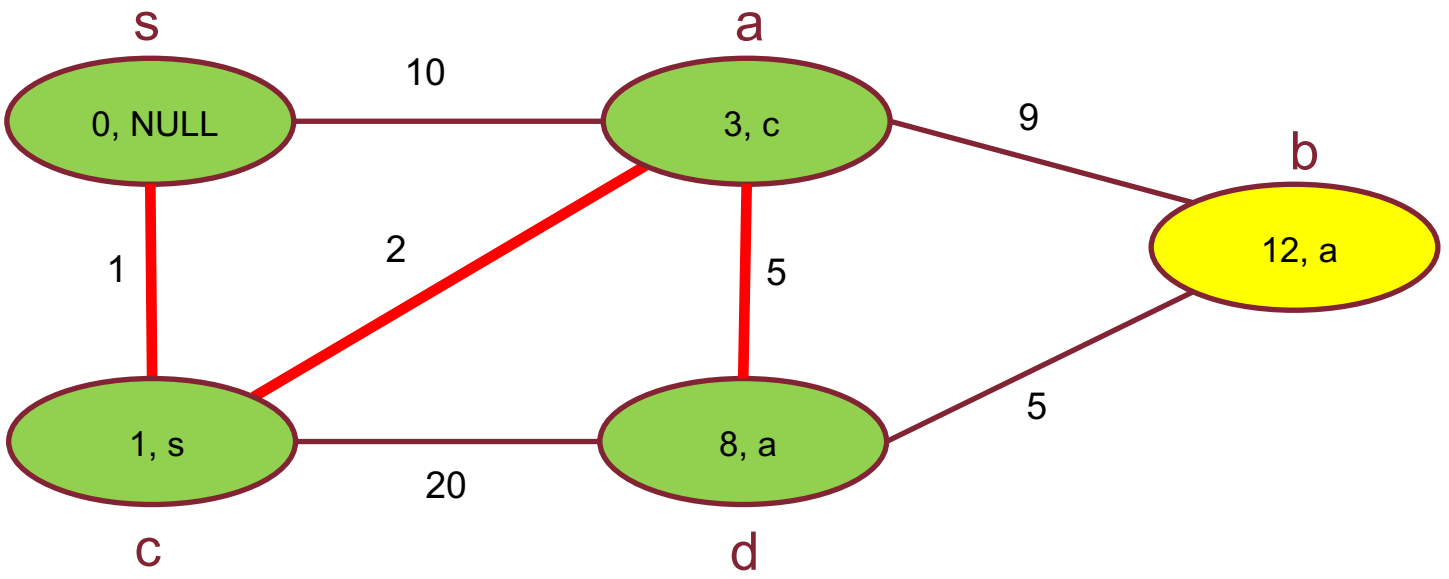


# Algoritmo di Dijkstra – esempio (7)

## Quarta iterazione

Nella quarta iterazione fra i vertici in F si sceglie quello a distanza minima (d), che viene spostato in VIS. Non c'è più alcun nuovo vertice da scoprire e l'aggiornamento non modifica i valori del vertice b dato che:

***dist(d)+peso arco (d,b)=8+5=13 > dist(b) = 12***



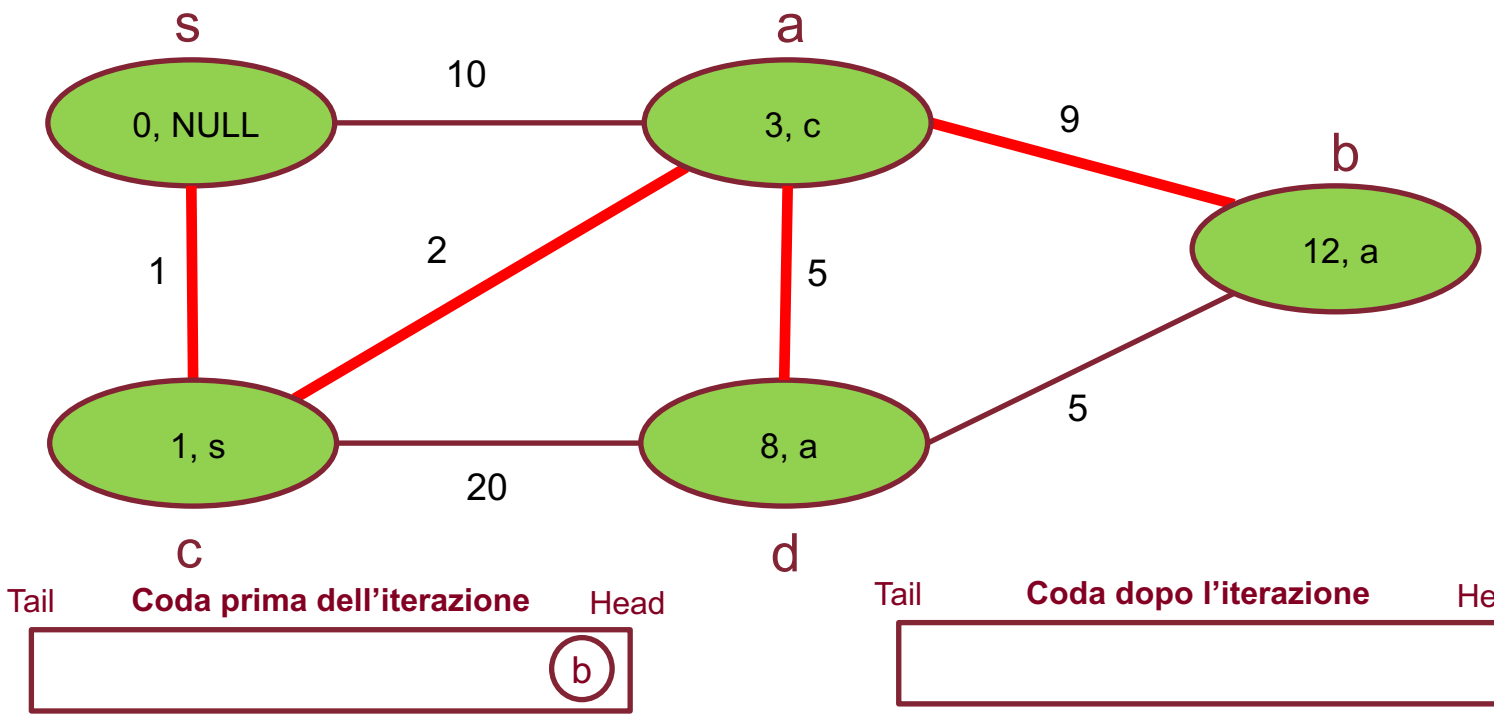

Tail **Coda prima dell'iterazione** Head


Tail **Coda dopo l'iterazione** Head

# Algoritmo di Dijkstra – esempio (6)

## Quinta ed ultima iterazione

Fra i vertici in F si sceglie quello a distanza minima (b), che è anche l'ultimo presente nella coda. Esso viene spostato in VIS e si termina. L'albero dei cammini minimi dalla sorgente s a tutti gli altri vertici è costituito dagli archi rossi.



# Algoritmo di Dijkstra – costo computaz. (1)

L'inizializzazione ha complessità $\Theta(|V|)$.

L'algoritmo compie successivamente $\Theta(|V|)$ iterazioni, dato che ad ogni iterazione si fissa il cammino minimo per un nuovo vertice (quello estratto dalla coda).

Nel complesso di tali iterazioni si effettuano:

- $\Theta(|V|)$ inserzioni nella coda (ogni vertice è inserito una sola volta);
- $\Theta(|V|)$ estrazioni dalla coda (ogni vertice è estratto una volta sola);
- $O(|E|)$ aggiornamenti di un vertice nella coda (un aggiornamento possibile per ogni arco del grafo).

# Algoritmo di Dijkstra – costo computaz. (2)

I costi delle varie operazioni sulla coda dipendono dalla sua implementazione.

Se la coda è implementata ad esempio tramite un vettore non ordinato:

- l'inserimento ha costo $\Theta(1)$;
- l'estrazione ha costo $O(|V|)$;
- l'aggiornamento di un vertice ha costo $\Theta(1)$.

Dunque in tal caso il costo computazionale risulta:

$$O(|V| + |V| + |V|^2 + |E|) = O(|V|^2 + |E|) = O(|V|^2).$$

# Algoritmo di Dijkstra – costo computaz. (3)

Viceversa, se la coda è implementata tramite uno heap:

- l'inserimento ha costo $O(\log|V|)$;
- l'estrazione ha costo $O(\log|V|)$;
- l'aggiornamento di un vertice ha costo $O(\log|V|)$.

Dunque in tal caso il costo computazionale risulta:

$$O(|V| + |V| \log|V| + |V| \log|V| + |E| \log|V|) =$$
$$= O((|V| + |E|) \log|V|).$$

# Algoritmo di Dijkstra – costo computaz. (4)

Si noti che il costo computazionale con lo heap può risultare minore o maggiore di quello con il vettore non ordinato a seconda che il grafo abbia pochi o tanti archi:

| | **Grafo sparso $\|E\| = O(\|V\|)$** | **Grafo denso $\|E\| = O(\|V\|^2)$** |
|---|---|---|
| Costo con lo heap | $O((\|V\| log \|V\|)$ | $O(\|V\|^2 log \|V\|)$ |
| Costo col vettore non ordinato | $O(\|V\|^2)$ | $O(\|V\|^2)$ |

# Algoritmo di Dijkstra – correttezza (1)

La correttezza dell'algoritmo di Dijkstra si dimostra per induzione sulla cardinalità dell'insieme VIS.

L'algoritmo è banalmente corretto quando:

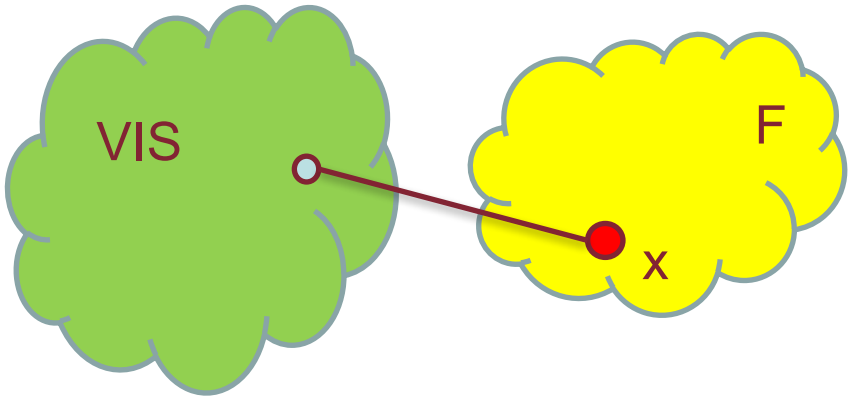
- $|VIS| = 1$: VIS contiene solo la sorgente, che è a distanza zero da se stessa.
- $|VIS| = 2$: VIS contiene solo la sorgente s ed il vertice collegato ad s *dall'arco di peso minimo fra quelli uscenti da s*.

# Algoritmo di Dijkstra – correttezza (2)

Sia $|VIS| = k$.

Il prossimo vertice che verrà inserito in VIS è un vertice x (in rosso nella figura), fra quelli in F, tale che:

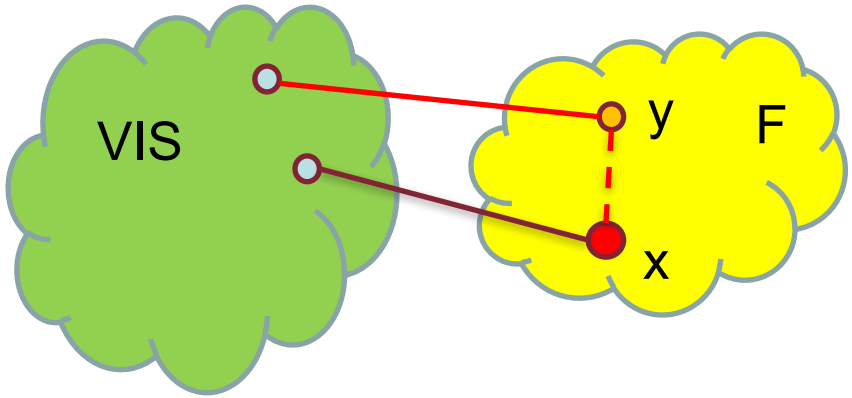
- x è raggiungibile direttamente da un vertice in VIS;
- dist(x) ha valore minimo fra i vertici in F.



# Algoritmo di Dijkstra – correttezza (2)

Se esistesse un cammino C che permette di raggiungere x con un peso complessivo ***strettamente minore*** di dist(x), esso dovrebbe necessariamente passare anche per altri vertici in F dato che passando per i soli vertici in VIS la distanza minima è appunto dist(x).
Ma tale cammino sarebbe costituito da una porzione in VIS, da un arco che porta da VIS a un vertice y in F (arancio nella figura), più una porzione in F (in rosso tratteggiato nella figura). Dato che dist(y) non può essere minore di dist(x) (altrimenti y sarebbe stato scelto al posto di x) ne segue che la porzione in F dell'ipotetico cammino C ***dovrebbe avere peso negativo***, il che contraddice l'ipotesi di partenza che tutti gli archi del grafo abbiano peso ≥ 0.

CVD



# *Archi peso negativo e cost. additive*

## ESERCIZIO 1

Sia $G = (V, E)$ un qualsiasi grafo orientato con pesi sugli archi, pesi che possono essere anche negativi ma in cui non sono presenti cicli di peso negativo.

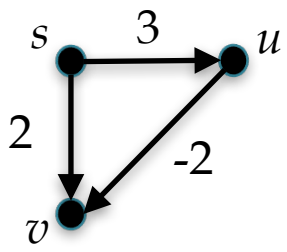
- Dimostrare che l'algoritmo di Dijkstra su grafi di questo tipo non calcola necessariamente i cammini di costo minimo tra la sorgente e gli altri nodi del grafo. [anche **Cormen, 24.4-2**]

- Per il calcolo dei cammini di costo minimo in $G$ si suggerisce il seguente algoritmo:
  *Sia $M$ il costo minimo tra i costi degli archi di $G$. Modifichiamo i pesi degli archi di $G$ sommando a ciascuno di questi l'intero $|M|$ abbastanza grande da renderli tutti positivi. Al grafo che si ottiene $G'$ (che ha pesi positivi) applichiamo l'algoritmo di Dijkstra.*
  I cammini minimi che vengono così calcolati sono anche cammini minimi per il grafo originale $G$? Motivare la risposta.

# *Archi di costo negativo: soluzione*

**1)** Perché l'algoritmo di Dijkstra fallisce con archi di peso negativo? Sostanzialmente perché via via seleziona i nodi più vicini alla radice (un po' come una BFS, ma generalizzando la nozione di vicinanza con i pesi sugli archi).

Tuttavia, un nodo che "sembrava" stabilizzato alla sua distanza minima potrebbe essere raggiunto da un cammino di costo minore a causa di archi con pesi negativi. E quindi ecco un esempio minimale:

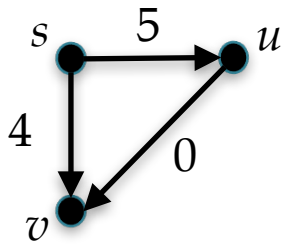



Al nodo *v* viene assegnata distanza 2, in quanto nodo più vicino a *s*, e viene ritenuto dall'algoritmo di Dijkstra "stabilizzato" al suo costo minimo. Infatti, con pesi positivi, nessun cammino che passa per *u* potrà essere migliore.

# Costanti additive: soluzione

**2)** Purtroppo, aggiungere una costante additiva M ai pesi di tutti gli archi non permette generalizzare l'algoritmo di Dijkstra. Il problema è che la costante additiva M **influenza il peso dei cammini in modo diverso** a seconda della lunghezza del cammino: $w'(p)=w(p) + |p| \cdot M$.

Il controesempio può essere facilmente derivato dall'esempio precedente, aggiungendo 2 per rendere non negativo l'arco $u \rightarrow v$. Questa costante penalizza comunque il cammino $s \rightarrow u \rightarrow v$ più del cammino $s \rightarrow v$.




Ritorneremo su questo punto in un esercizio successivo.

# *Lezione 25*

# *That's all Folks…*

# *…Domande?*

Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni in modalità mista o a distanza

# Cenni al funzionamento e storia di Internet

**Giancarlo Bongiovanni**
**Ivano Salvo**


SAPIENZA
UNIVERSITÀ DI ROMA

Queste dispense sono state realizzate sulla base delle slide preparate da T. Calamoneri e G. Bongiovanni per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

# Breve storia di Internet (1)

La attuale rete Internet è figlia della rete Arpanet, nata nei primi anni ‘60 come risultato delle ricerche finanziate dall’Advanced Research Project Agency (ARPA) del DoD (Department of Defense, USA).

Si era nel pieno della guerra fredda, il DoD voleva una rete di trasmissione dati che potesse continuare a funzionare, almeno parzialmente, anche in caso di catastrofe nucleare.

Il risultato delle ricerche finanziate dal DoD fu la rete Arpanet, una ***rete a commutazione di pacchetto***, la cui evoluzione è l’odierna rete Internet.

# Commutazione di circuito vs Commutazione di pacchetto (1)

Fino agli anni ‘60 lo sviluppo di reti di trasmissione dati era dominato da una mentalità di progetto «telefonica», ossia derivante dalla formazione culturale e dall’esperienza maturate nello sviluppo delle reti di fonia (per la trasmissione della voce).

L’idea era la seguente:

- i terminali (gli apparecchi utilizzati dagli utenti finali) non possedevano capacità di elaborazione;
- le capacità di elaborazione erano fornite integralmente dall’infrastruttura della rete di fonia (ossia dalle centrali telefoniche), che provvedevano a connettere i terminali fra loro per consentire la comunicazione.

# Commutazione di circuito vs Commutazione di pacchetto (2)

Quando due utenti volevano comunicare, l'infrastruttura della rete di fonia doveva prima di tutto creare un *collegamento elettrico* diretto fra loro, detto ***circuito***.

Solo dopo che tale circuito era stato stabilito i due utenti potevano iniziare a scambiarsi informazioni (ossia: a parlare).

Quando i due utenti terminavano la conversazione, il circuito veniva interrotto.

Questa operatività ha dato origine al termine ***commutazione di circuito***.

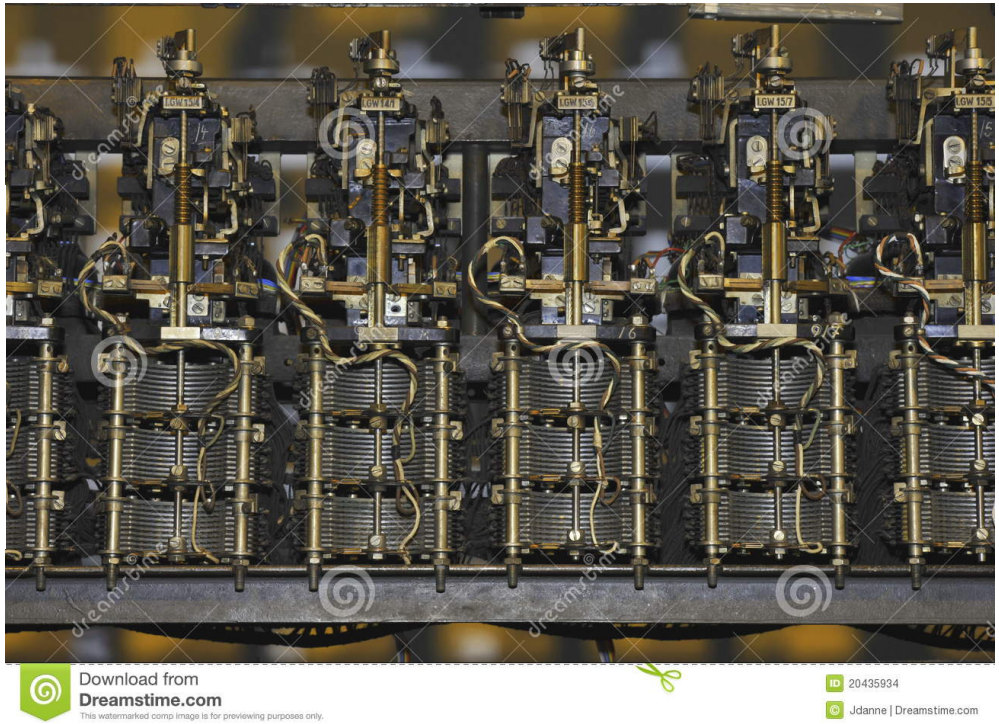
# Commutazione di circuito vs Commutazione di pacchetto (3)

Fine ‘800: l’operatrice/operatore in centrale stabilisce il *circuito elettrico* fra i due utenti:



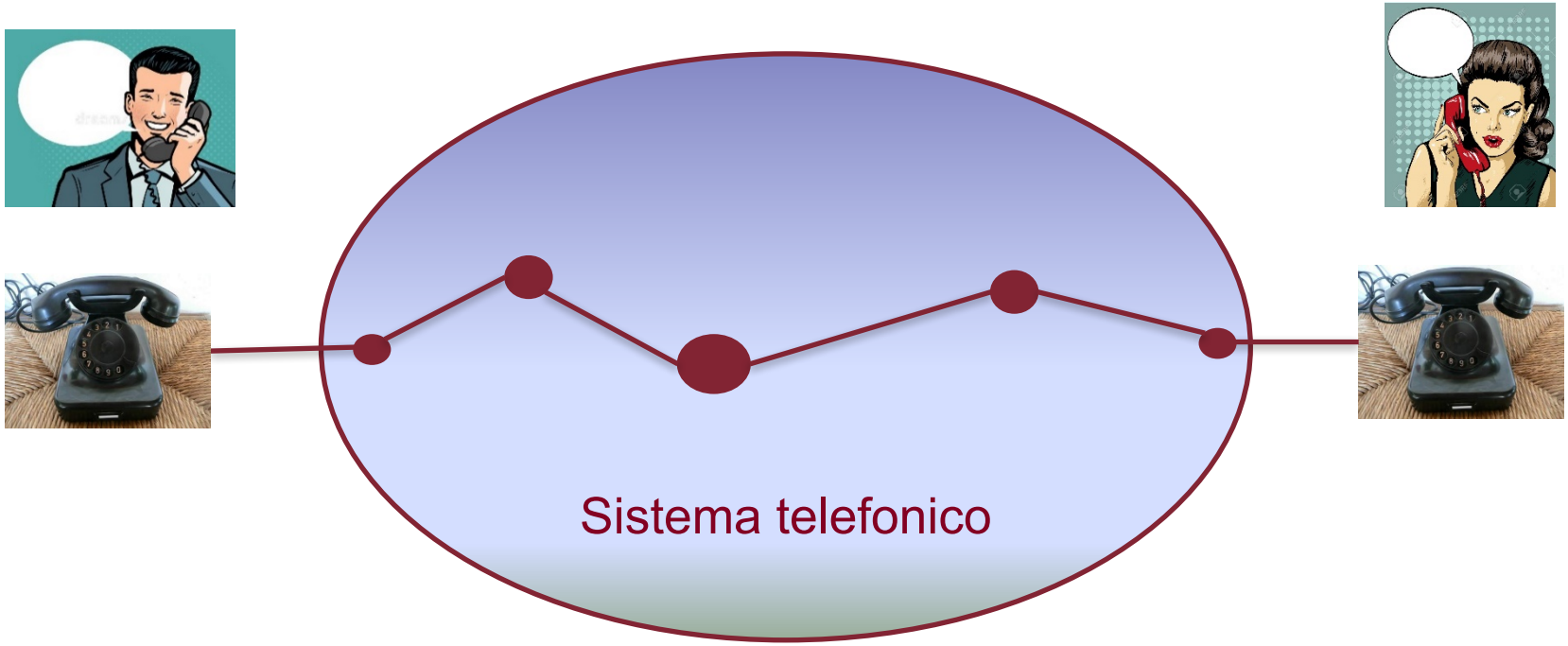
# Commutazione di circuito vs Commutazione di pacchetto (4)

Dagli anni ‘20 (Strowger switch, brevetto di fine ‘800): centraline elettromeccaniche mettono automaticamente in *collegamento elettrico* gli utenti:




# Commutazione di circuito vs Commutazione di pacchetto (5)

Esempio di commutazione di circuito



# Commutazione di circuito vs Commutazione di pacchetto (6)

L'approccio alla base di Arpanet (e, oggi, di Internet) è diametralmente opposto.

L'idea è la seguente:

- i terminali (gli apparecchi utilizzati dagli utenti finali) possiedono capacità di elaborazione;
- l'infrastruttura della rete dati, che provvede a connettere i terminali fra loro, fornisce invece un *servizio minimale*, consistente nel solo trasporto di ***pacchetti*** di dati dalla sorgente alla destinazione, senza garantire né la consegna né il corretto ordine d'arrivo;
- I terminali, grazie alla loro capacità di elaborazione, provvedono a risistemate le cose (riordino, ritrasmissione, ecc.)
- questo tipo di funzionamento prende il nome di ***commutazione di pacchetto***.

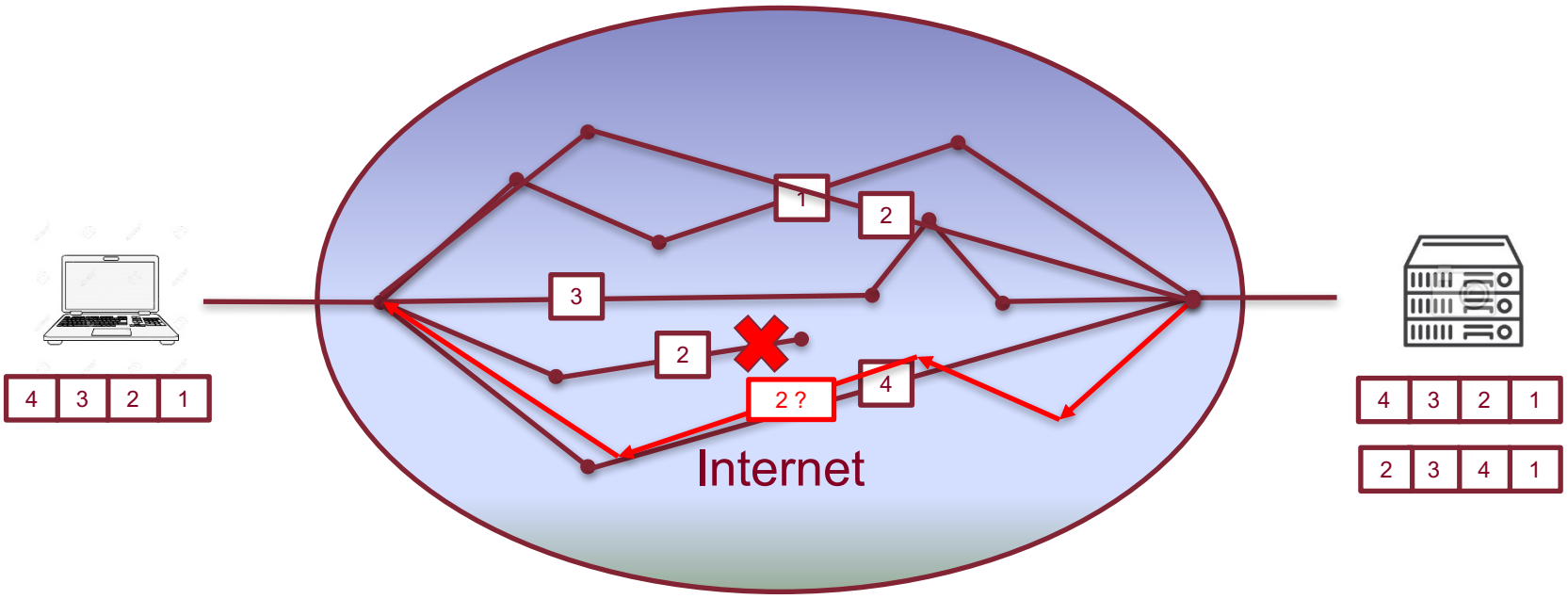
# Commutazione di circuito vs Commutazione di pacchetto (7)

Aspetti importanti:

- Ogni pacchetto viaggia nella rete dati indipendentemente dagli altri e può seguire un percorso diverso dagli altri.
- Può essere danneggiato o perso durante il transito.
- I terminali sono preparati alla perdita di pacchetti e sanno come reagire: hanno gli strumenti per individuare i pacchetti mancanti e richiederne la ritrasmissione.
- Anche per quanto riguarda l'arrivo di pacchetti in un ordine non corretto i terminali sono in grado di rimediare, rimettendoli nella giusta sequenza.

# Commutazione di circuito vs Commutazione di pacchetto (8)

Esempio di commutazione di pacchetto



# Commutazione di circuito vs Commutazione di pacchetto (9)

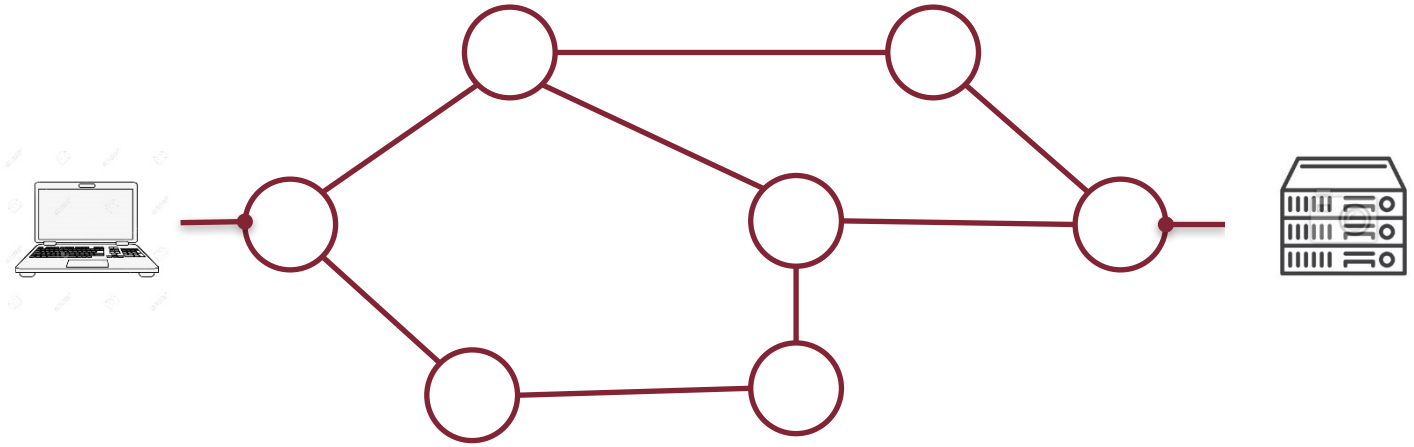
Affidabilità:

- Nella commutazione di circuito, se cade una parte del sistema telefonico tutti i circuiti che usano tale porzione vengono interrotti e devono essere ricreati ex novo.
- Nella commutazione di pacchetto, se cade una parte della rete si perdono solo i pacchetti in transito in quella parte. Essi vengono ritrasmessi (e seguono altre strade) non appena i terminali si accorgono del loro mancato arrivo e segnalano il problema.

# Funzionamento di Internet (1)

La componente di Internet che permette la trasmissione di dati fra terminali remoti si chiama ***Subnet di comunicazione*** ed è costituita da:

- ***router*** (o *sistemi intermedi*, o *apparati store-and-forward*)
- ***linee di trasmissione punto a punto***, che connettono ciascuna una coppia di router:



# Funzionamento di Internet (2)

I router di Internet sono centinaia di migliaia e ognuno funziona secondo questo principio:

- ogni router riceve i pacchetti dalle sue linee di ingresso e li smista sulle linee di uscita;
- ogni pacchetto ricevuto:
  - viene memorizzato in un buffer;
  - viene instradato sulla opportuna linea di uscita che costituisce la migliore soluzione per far arrivare il pacchetto a destinazione;
  - la scelta della linea da usare avviene consultando una apposita ***tabella di instradamento***, che viene aggiornata con grande frequenza (ogni pochi minuti).

# Funzionamento di Internet (3)

Le linee di trasmissione punto-a-punto sono caratterizzate da:

- ***banda trasmissiva***, misurata in bit/secondo: varia da alcuni megabit/sec. a decine di gigabit/sec;
- ***tempo di propagazione*** del segnale da un capo all'altro della linea (ridotto per le linee terrestri, elevato per le linee satellitari);
- ***costo*** di attraversamento, strettamente positivo, calcolato combinando le caratteristiche di cui sopra con ulteriori dati (ad esempio, il tempo stimato di accodamento dei pacchetti nel router all'altro capo della linea).

# Funzionamento di Internet (4)

Di conseguenza la subnet di comunicazione può essere modellata come un grafo pesato, nel quale:

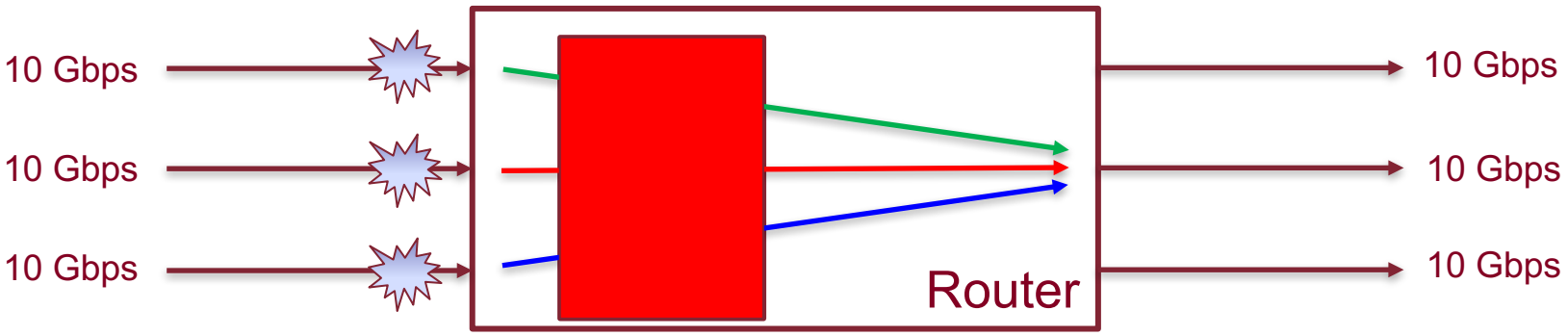
- i vertici corrispondono ai router;
- gli archi corrispondono alle linee di trasmissione;
- il peso di ogni arco corrisponde al costo di attraversamento della linea in quel momento.

Viene quindi naturale l'idea di trovare, per ciascun router, l'insieme dei cammini di costo minimo per raggiungere tutti gli altri router (ossia tutti gli altri vertici del grafo).

# Funzionamento di Internet (5)

Il grafo può cambiare molto rapidamente, per via di:

- malfunzionamenti di router;
- cadute di linee di trasmissione;
- insorgere di congestione, fenomeno che deriva da improvvisi e concentrati picchi di traffico dati e che non è prevedibile in anticipo:




Di conseguenza il calcolo dei cammini di costo minimo va eseguito con grande frequenza.

# Algoritmo di Dijkstra (1)

Esistono vari algoritmi per la ricerca dei cammini di peso minimo, noi vedremo l'algoritmo di Dijkstra (1956).

Dato un ***grafo orientato e pesato G*** con ***pesi degli archi non negativi*** ed uno specifico vertice di partenza ***s***, l'algoritmo di Dijkstra si muove sistematicamente lungo gli archi di G per costruire un ***albero dei cammini minimi*** (che inizialmente consiste del solo vertice s, la sua radice) che, alla fine, contiene tutti i vertici raggiungibili da s. Per ogni vertice v raggiungibile da s il cammino da s a v nell'albero dei cammini minimi è un cammino di peso minimo da s a v, ossia un cammino che ha la seguente proprietà:

- la somma dei pesi degli archi che costituiscono il cammino è minore o uguale alla somma dei pesi degli archi di qualunque altro cammino da s a v.