Medicina e Tecnologie TD
Lezione 10

Pierangelo Veltri
pierangelo.veltri@unical.it

Definizioni

Sia $G=(V,E)$ un grafo orientato con costi $w(v_i,v_j)$ sugli archi. Il costo di un cammino $\pi=<v_0,v_1,v_2,\dots,v_k>$ è dato da:

$$w(\pi)=\sum_{i=1}^{k} w(v_{i-1},v_i)$$

Un cammino minimo tra una coppia di vertici x e y è un cammino di costo minore o uguale a quello di ogni altro cammino tra gli stessi vertici.

Proprietà dei cammini minimi

- Sottostruttura ottima: ogni sottocammino di un cammino minimo è anch'esso minimo
- Grafi con cicli negativi: se due vertici x e y appartengono a un ciclo di costo negativo, non esiste nessun cammino minimo finito tra di essi
- Se G non contiene cicli negativi, tra ogni coppia di vertici connessi in G esiste sempre un cammino minimo semplice, in cui cioè tutti i vertici sono distinti

Distanza fra vertici

- La distanza d_{xy} tra due vertici x e y è il costo di un cammino minimo tra da x a y, o $+\infty$ se i due vertici non sono connessi
- Disuguaglianza triangolare: per ogni x, y e z

$$d_{xz} \leq d_{xy} + d_{yz}$$

- Condizione di Bellman: per ogni arco (u,v) e per ogni vertice s

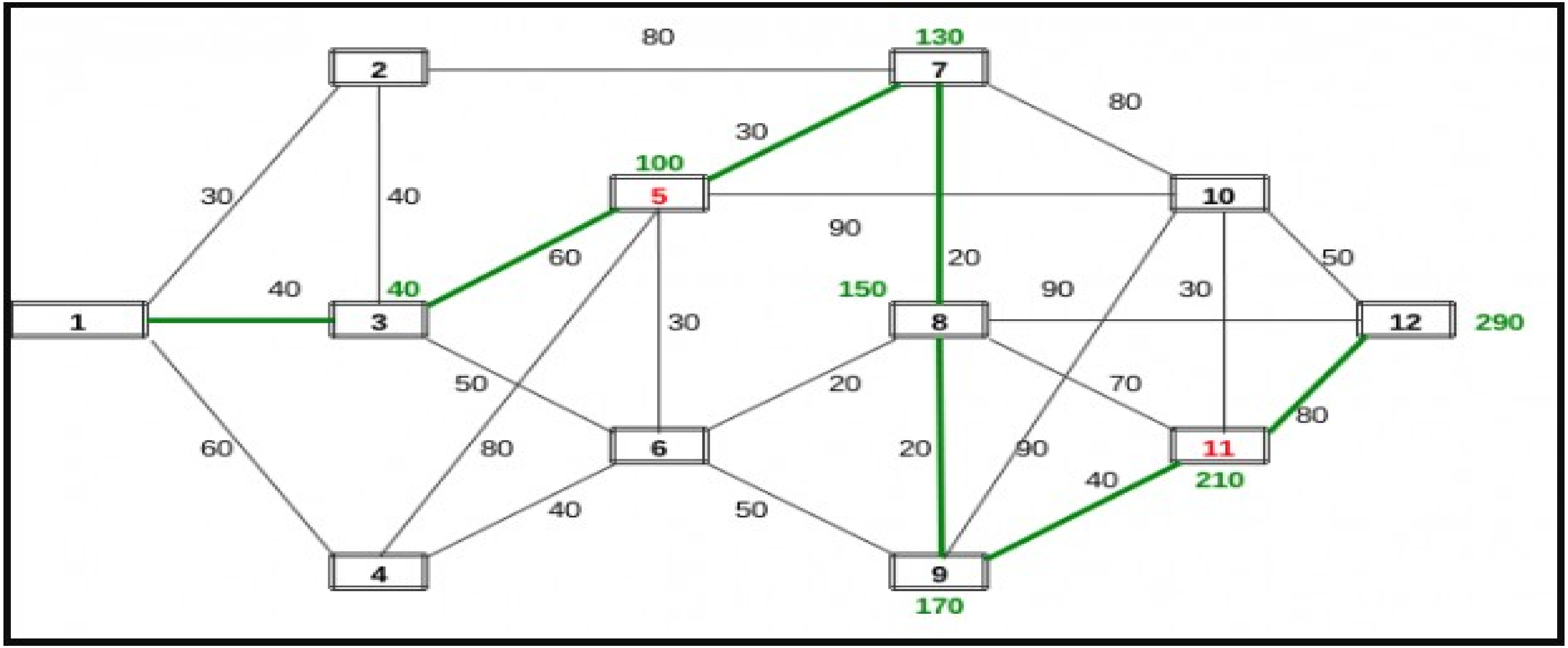
$$d_{su} + w(u, v) \geq d_{sv}$$

Alberi di cammini minimi

- Un arco (u,v) appartiene a un cammino minimo a partire da un vertice s se e solo se u è raggiungibile da s e $d_{su}+w(u,v)=d_{sv}$

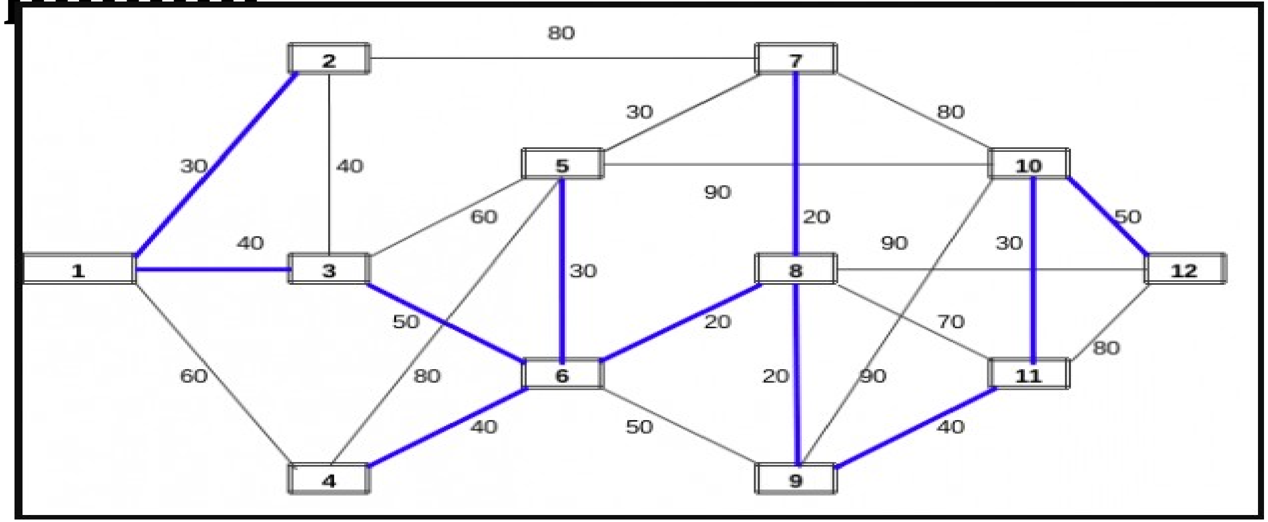
- I cammini minimi da un vertice s a tutti gli altri vertici del grafo possono essere rappresentati tramite un albero radicato in s, detto albero dei cammini minimi

Cammini Minimi



Albero dei Cammini Minimi

I cammini minimi da un vertice s a tutti gli altri vertici del grafo possono essere rappresentati tramite un albero la cui radice è s, detto **albero dei cammini minimi**



- Programmazione dinamica: l’algoritmo di Floyd-Warshall

L’algoritmo di Floyd-Warshall permette di calcolare, dato un grafo con pesi non negativi, le distanze minime tra tutte le coppie di nodi.

```python
import GraphL as g

def Floyd(G):
    C = g.copyGraph(G)
    for i in g.nodes(C):
        g.insertEdge(C,i,i,0)
    for k in g.nodes(C):
        for i in g.nodes(C):
            for j in g.nodes(C):
                w = g.weight(C,i,k) + g.weight(C,k,j)
                if w <  g.weight(C,i,j):
                    g.insertEdge(C,i,j,w)
    return C
```

Listing 11.1: Algoritmo di Floyd

Floyd-Warshall

- Algo floyd-warshall
- Consideriamo i verici intermedi di un cammino minimo ossia:
  - Sia p=<v1,….vl> un cammino, un vertice intermedio di p è un qualsiasi vertice diverso da v1 e vl.
  - Sia V {1…n} l insieme dei vertici di G e sia {1…k} un sottoinsieme di vertici per k qualsiiasi . Per ogni coppia di vertici i,j si considerino tutti i cammini da i a j i cui vertici intermedi stanno in 1..k e sia p un cammino di peso minimo tra di essi (si assume che G non abbia cicli di peso negativo)
  - L algo sfrutta una **relazione** tra il cammino minimo p ed i cammini minimi dai a j aventi tutti i vertici intermedi nell’insieme {1…k-1}. **La relazione cambia a seconda se k sia un vertice intermedio del cammino oppure no**

Floyd-Warshall

- Se k non è un vertice intermedio del cammino p, allora tutti i vertici intermedi del cammino p sono nell’insieme 1 .. k’-1. quindi un cammino minimo da i a j on tutti i vertici intermedi in 1..k-1 è minimo da i a j con 1…k vertici intermedi
- Se invece k e’ un vertice intermedio nel cammino p allora dividiamo p in i-□k(passando per p1) e k-□j (passando per p2)
- P1 e’ minimo da i a k passando per 1..k-1. analogamente p2 e’ minimo da k a j con vertici in 1..k-1

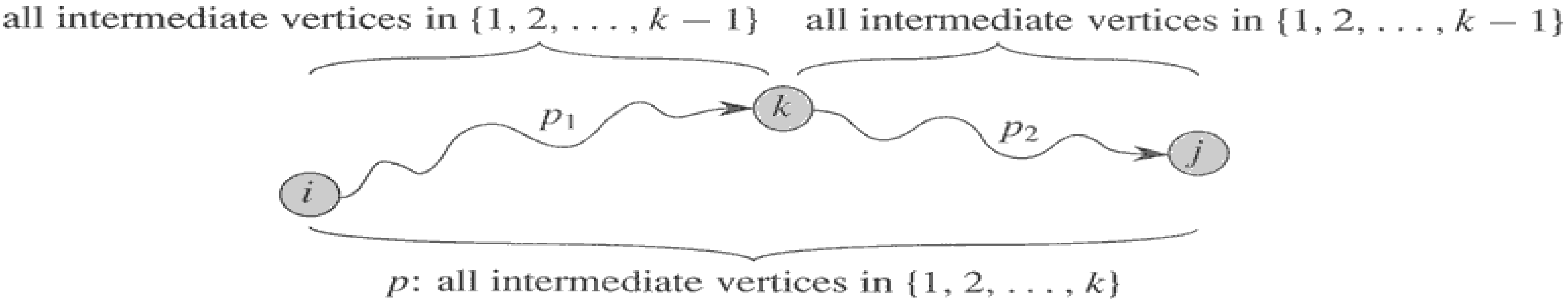


Figure 25.3 Path p is a shortest path from vertex i to vertex j, and k is the highest-numbered intermediate vertex of p. Path p_1, the portion of path p from vertex i to vertex k, has all intermediate vertices in the set $\{1, 2, \ldots, k-1\}$. The same holds for path p_2 from vertex k to vertex j.

- Sia d(i,j)k il il peso da un vertic i a j con 1…k vertici intermedi
- Sia W la matrice dei cammini minimi ossia w(ij) contiene il peso del cammino minimo da i a j

Dij(k) = wij se k=0

Altrimeni min $(d(ij)^{k-1}$, dik (k-1) + dkj (k-1)) se k >=1

- Sfruttando la definizione precedente

FLOYD-WARSHALL(W)

$n \leftarrow rows[W]$

$D^{(0)} \leftarrow W$

for $k \leftarrow 1$ **to** n

  do for $i \leftarrow 1$ **to** n

    do for $j \leftarrow 1$ **to** n

      do $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$

return $D^{(n)}$

Esempio di applicazione

- Ho un grafo e per ogni coppia

  di nodi voglio trovare il cammino minimo

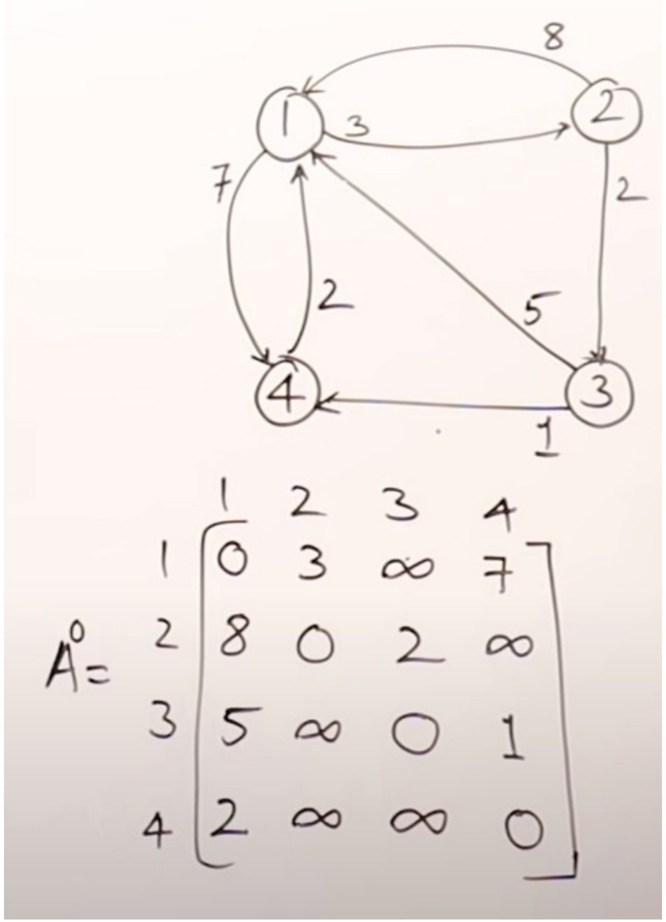
- Se parto da nodo 1 voglio il cammino minimo per

  1-2; 1-3; 1-4;

- Djakstra (vedremo) calcola il cammino minimo tra

  2 nodi (usato dai navigatori)

  M e’ n2 per cui nel caso naive e’ n2*n



Si risolve applicando la programmazione dinamica

- Ovvero cerco di usare decisioni intermedie
- Per esempio: se devo trovare un path a costo minimo tra 1 e 2 :
  - Posso avere un link diretto
  - O un link passando per il nodo 3 (1,3,2)
  - Oppure passando dal nodo 4 (1,4,2)

  cerco prima i cammini minimi dal vertice 1 e sfrutto risultati intermedi

- Preparo le sequenze considerando matrici intermedie di cammini
- La matrice iniziale considera il costo del cammino diretto e il costo pari ad infinito se due nodi non sono connessi
- Calcolo la matrice A1 in cui lascio i path del nodo 1

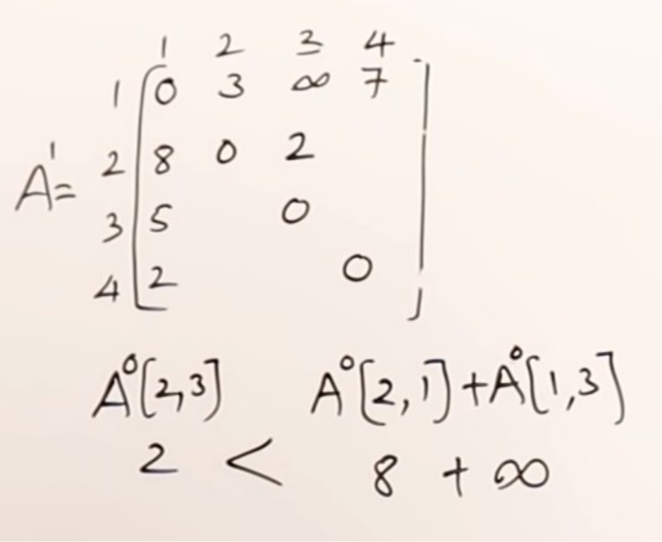
$$A^0 = \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$A^1 = \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & & \\ 5 & & 0 & \\ 2 & & & 0 \end{bmatrix} \end{array}$$

- Si costruisce la matrice A1 cercando i cammini con nodo intermedio

  cerco un cammino minimo da 2 a 3. Dalla matrice precedente
- Il costo e’ 2, cerco utilizzando cammini intermedi

  se il valore trovato e’ piu piccolo allora e’ quello altrimenti
- Resta il valore trovato



-

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & \infty & 0 \end{bmatrix} \end{array}$$

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & \\ & 0 & 2 & \\ 5 & 8 & 0 & 1 \\ & & 7 & 0 \end{bmatrix} \end{array}$$

$$A^2[1,2] \quad A^2[1,3] + A^2[3,2]$$

$$3 < 5 + 8$$

```
for(i=1; i<=n; i++)
{
    for(j=1;j<=n;j++)
    {
        A[i,j]=min(A[i,j],A[i,k]+A[k,j]);
```

```
for(k=1; k<=n; k++)
{
    for(i=1; i<=n; i++)
    {
        for(j=1;j<=n;j++)
        {
            A[i,j]=min(A[i,j],A[i,k]+A[k,j]);
        }
```

Vediamo il codice di riferimento per l’algoritmo

- Code

  https://www.programiz.com/dsa/floyd-warshall-algorithm

Algoritmi Greedy

- Un algoritmo greedy è un paradigma algoritmico, dove l'algoritmo cerca una soluzione ammissibile da un punto di vista globale attraverso la scelta della soluzione più appetibile (definita in precedenza dal programmatore) per quel determinato programma a ogni passo locale.

- Quando applicabili, questi algoritmi consentono di trovare soluzioni ottimali per determinati problemi in un tempo polinomiale, mentre negli altri non è garantita la convergenza all'ottimo globale.

- In particolare questi algoritmi cercano di mantenere una proprietà di sottostruttura ottima, quindi cercano di risolvere i sottoproblemi in maniera "avida" (da cui la traduzione letterale algoritmi avidi in italiano) considerando una parte definita migliore nell'input per risolvere tutti i problemi.

Algoritmo di Dijkstra

- L'algoritmo di Dijkstra è un algoritmo utilizzato per cercare i cammini minimi in un grafo con o senza ordinamento, **ciclico** e con **pesi non negativi** sugli archi.

- Fu inventato nel 1956 dall'informatico olandese Edsger Dijkstra che lo pubblicò successivamente nel 1959.

- Tale algoritmo trova applicazione in molteplici contesti quale l'ottimizzazione nella realizzazione di reti (idriche, telecomunicazioni, stradali, circuitali, ecc.) o l'organizzazione e la valutazione di percorsi runtime nel campo della robotica.

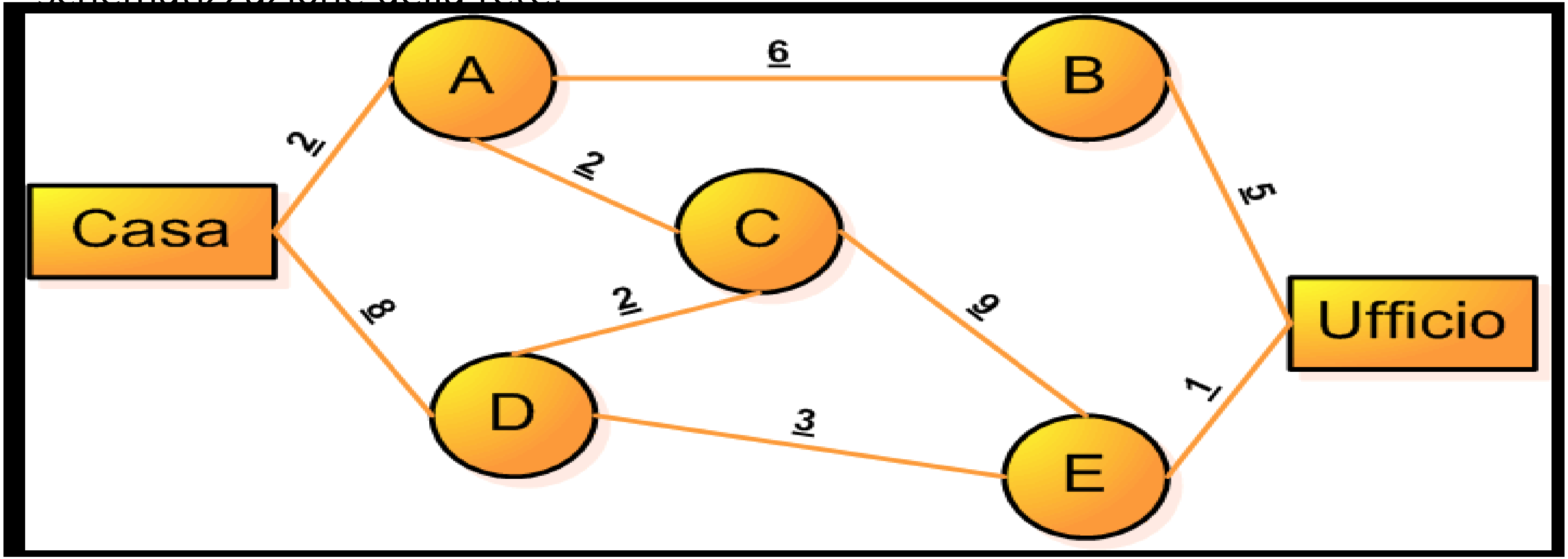
Algoritmo di Dijkstra

- Alla base di questi problemi c'è lo scopo di trovare il percorso minimo (più corto, più veloce, più economico...) tra due punti, uno di partenza e uno di arrivo

- è possibile ottenere non solo il percorso minimo tra un punto di partenza e uno di arrivo ma l'albero dei cammini minimi, *cioè tutti i percorsi minimi tra un punto di partenza e tutti gli altri punti della rete*.

Algoritmo di Dijkstra

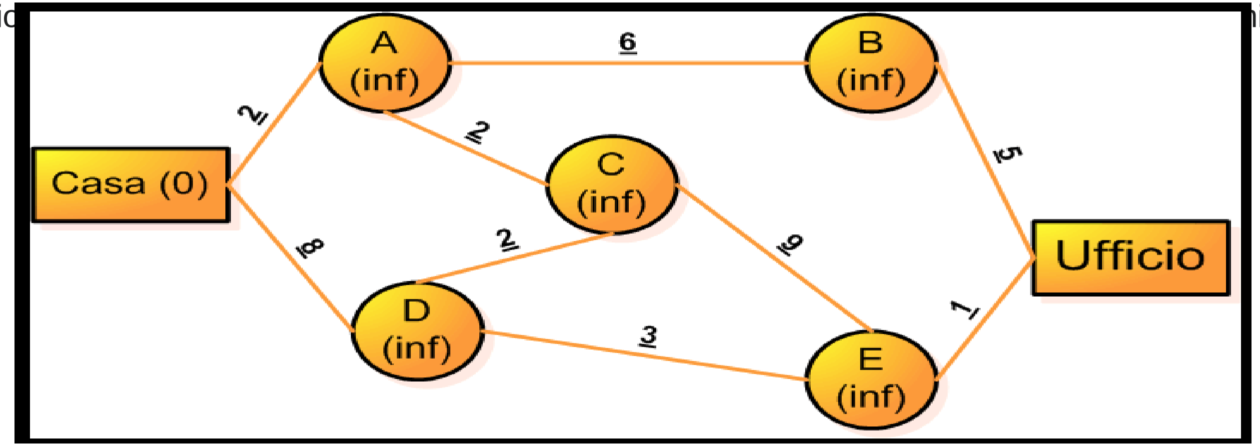
- Si consideri un problema in cui si vuole calcolare il percorso minimo tra casa e il posto di lavoro. Si schematizzino tutti i possibili percorsi e il relativo tempo di percorrenza (supponendo di voler calcolare il percorso più breve in fatto di tempo di percorrenza). I nodi A, B, C, D, E indicano le cittadine per cui è possibile passare. Ecco una schematizzazione della rete:



Algoritmo di Dijkstra

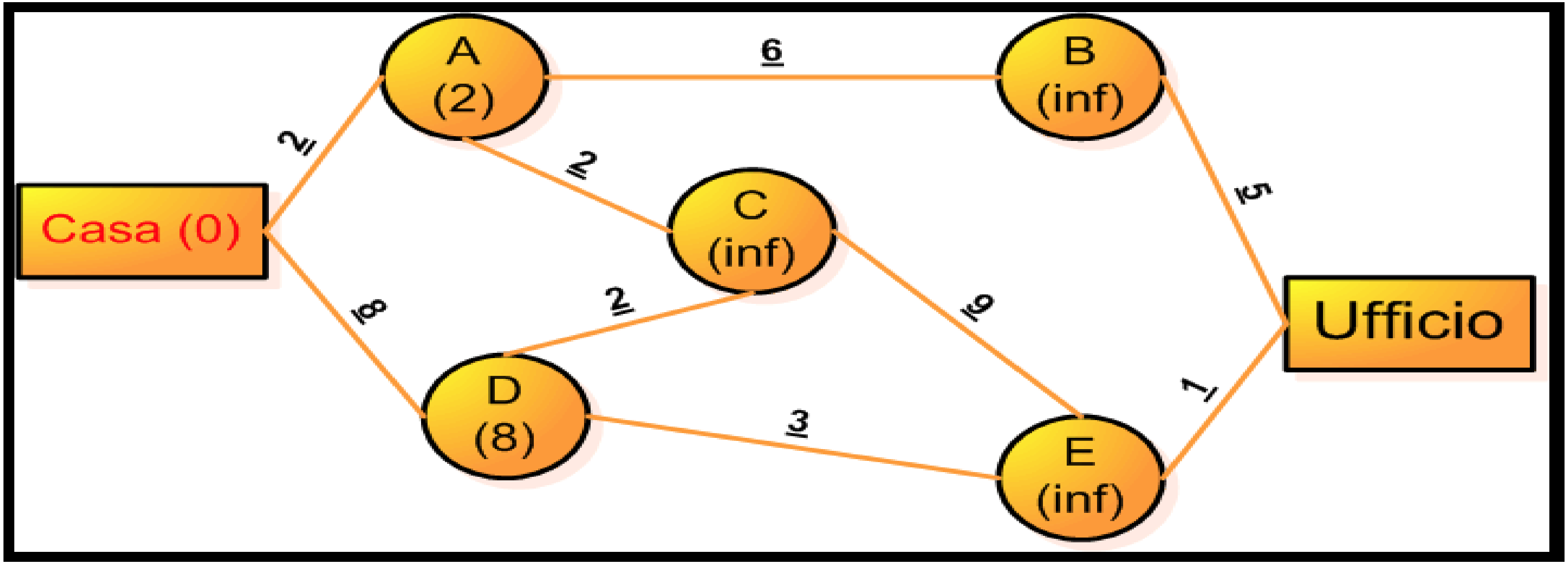
Bisogna ora assegnare a ogni nodo un valore, chiamato “potenziale”, seguendo alcune regole:

- Ogni nodo ha potenziale inizialmente pari ad infinito
- Il nodo di partenza (in questo caso “casa”) ha potenziale 0 (ovvero dista zero da sé stesso);
- Ogni volta si sceglie il nodo con potenziale minore e lo si rende definitivo (colorando il potenziale di rosso) e si aggiornano i nodi adiacenti;
- Il potenziale di un nodo è dato dalla somma del potenziale del nodo precedente + il costo del collegamento;
- Non si aggiornano i potenziali dei nodi resi definitivi;
- I potenziali definitivi indicano la distanza di quel nodo da quello di partenza;
- Quando si aggio … inimo).



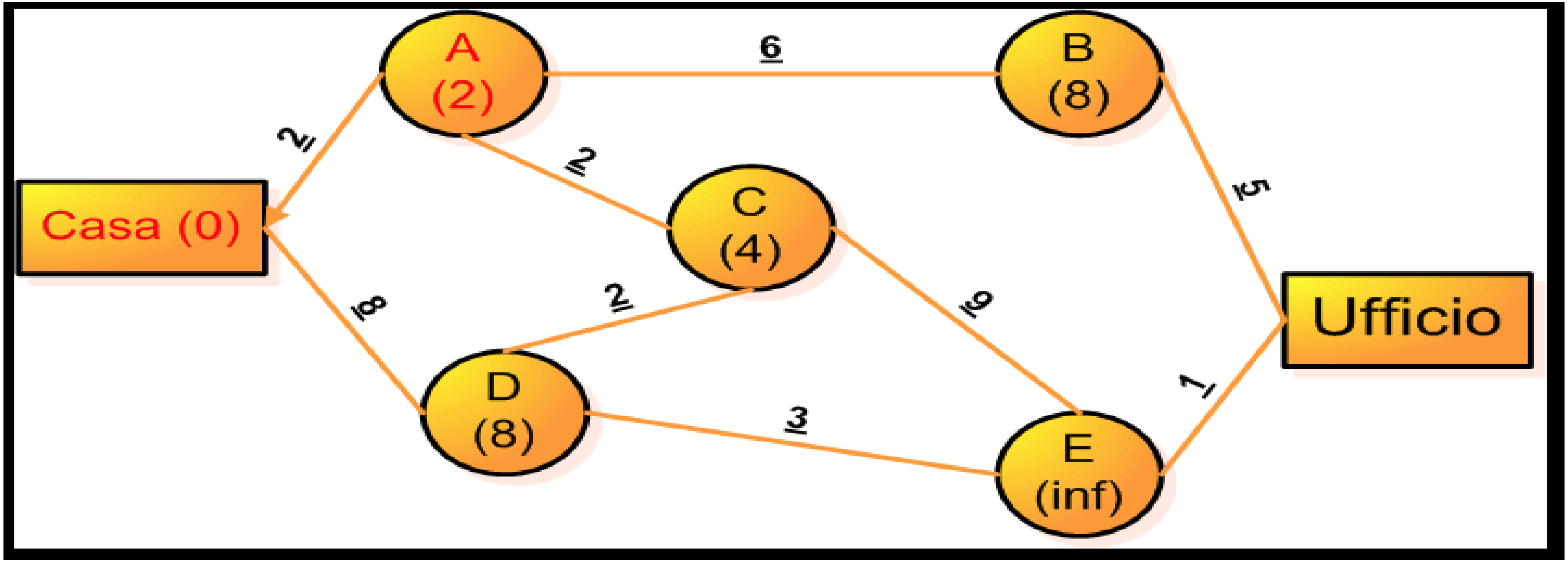
Algoritmo di Dijkstra

Seguendo le regole appena fissate si consideri il nodo con potenziale minore (“casa”) e lo si renda definitivo (colorandolo di rosso) e si aggiornino tutti i nodi adiacenti sommando l'attuale valore del potenziale (ovvero zero) al costo del percorso. Si aggiornino i potenziali perché si aveva, nel caso di A, potenziale infinito mentre ora il potenziale è 2. Ricordando che il potenziale minore è sempre preferibile. Ecco come si è aggiornata la rete:



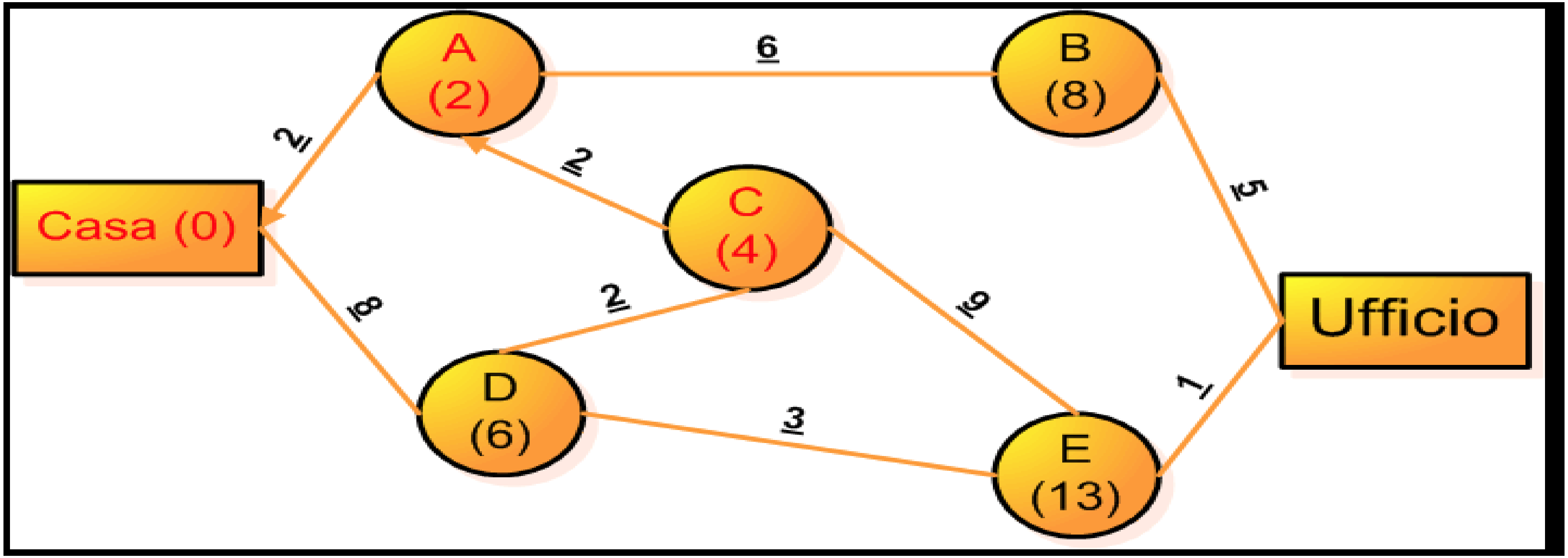
Algoritmo di Dijkstra

Bisogna ora considerare il nodo non definitivo (ovvero quelli scritti in nero) con potenziale minore (il nodo A). Lo si rende definitivo e si aggiornano i potenziali dei nodi adiacenti B e C. Si indichi con una freccia da dove proviene il potenziale dei nodi resi definitivi.



Algoritmo di Dijkstra

Il nodo con potenziale minore ora è C. lo si rende definitivo e si aggiornano quelli adiacenti.

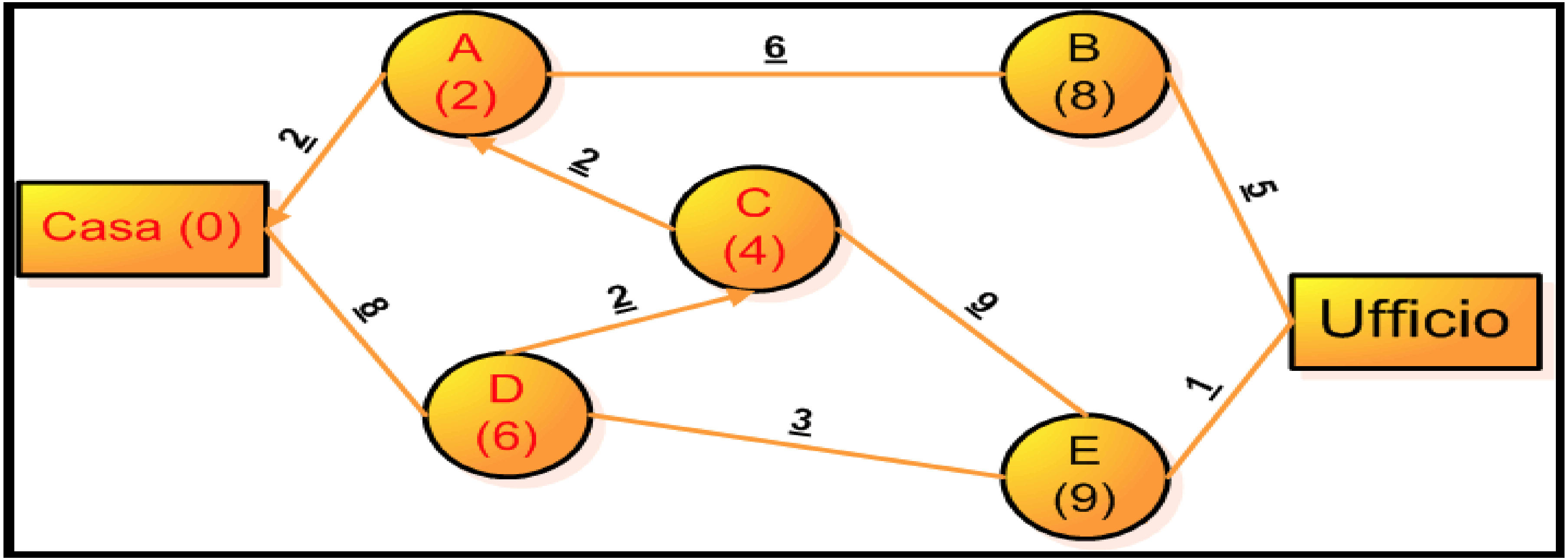


Algoritmo di Dijkstra

Va notato come il nodo D abbia ora potenziale 6 in quanto 6 è minore di 8 e quindi lo si aggiorna.

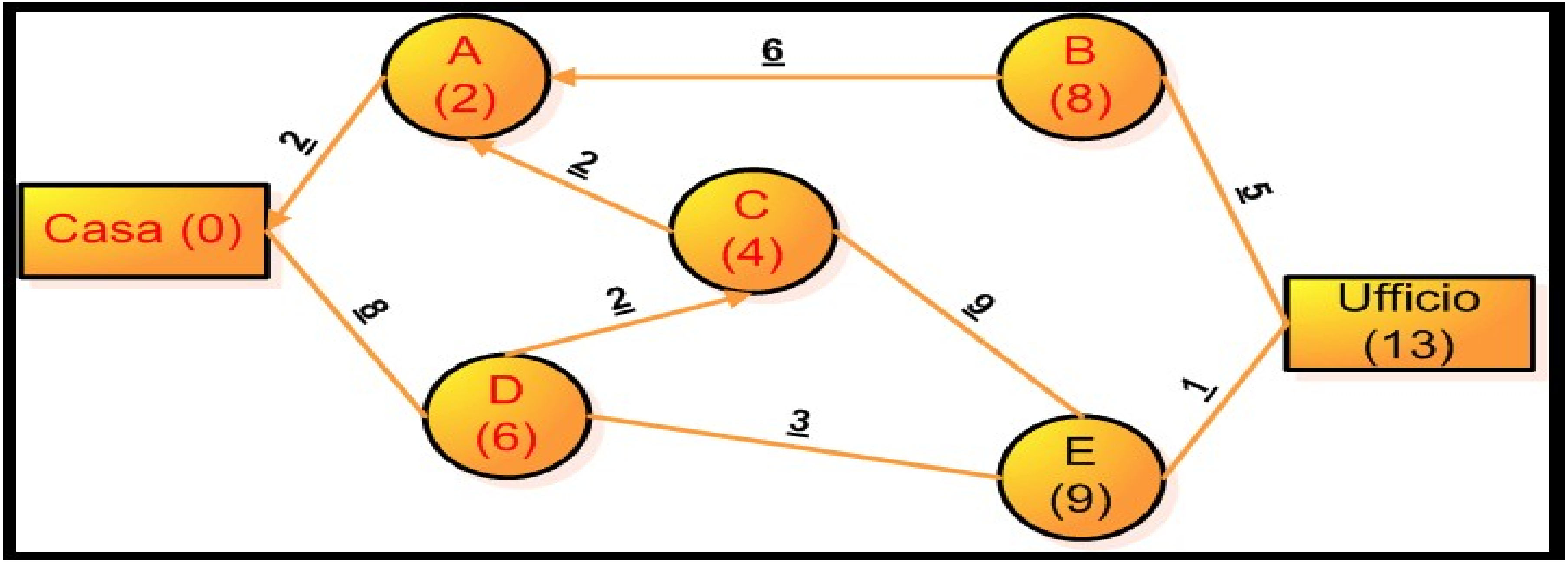
Se si fosse ottenuto un valore maggiore di quello che già c'era si sarebbe dovuto lasciare invariato.

Si renda definitivo il nodo D e si aggiorni il grafico:



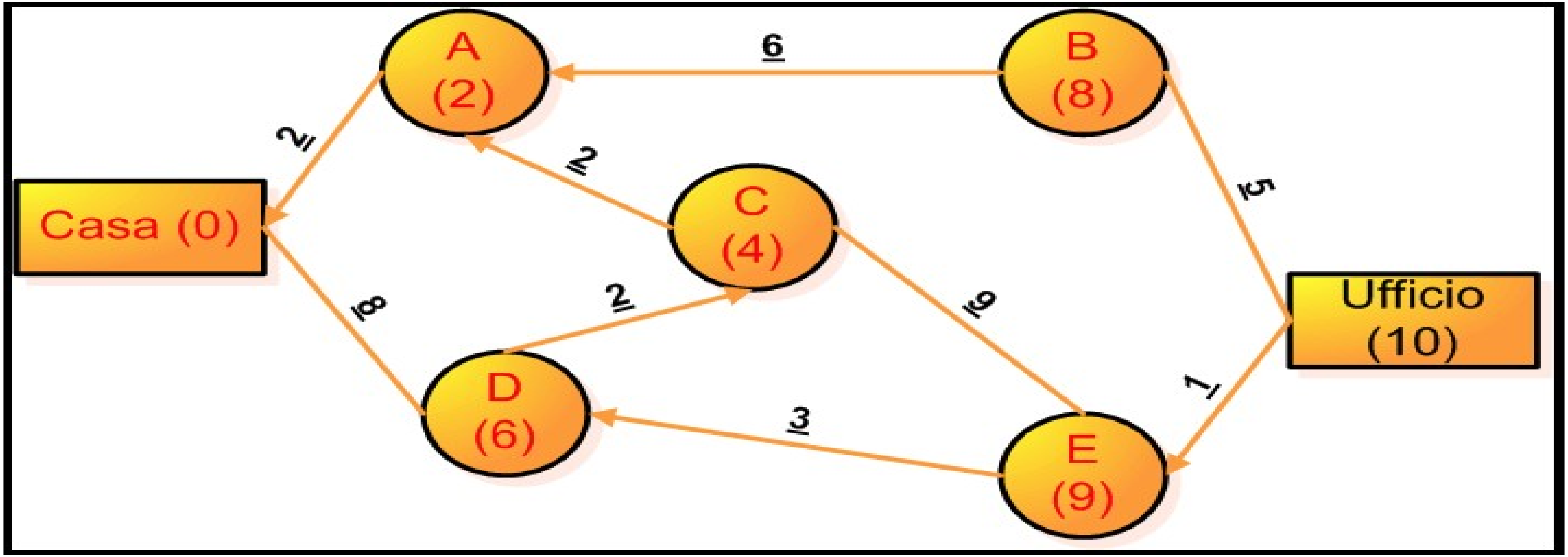
Algoritmo di Dijkstra

Il nodo con potenziale minore restante è B e lo si rende definitivo aggiornando di conseguenza il grafico:



Algoritmo di Dijkstra

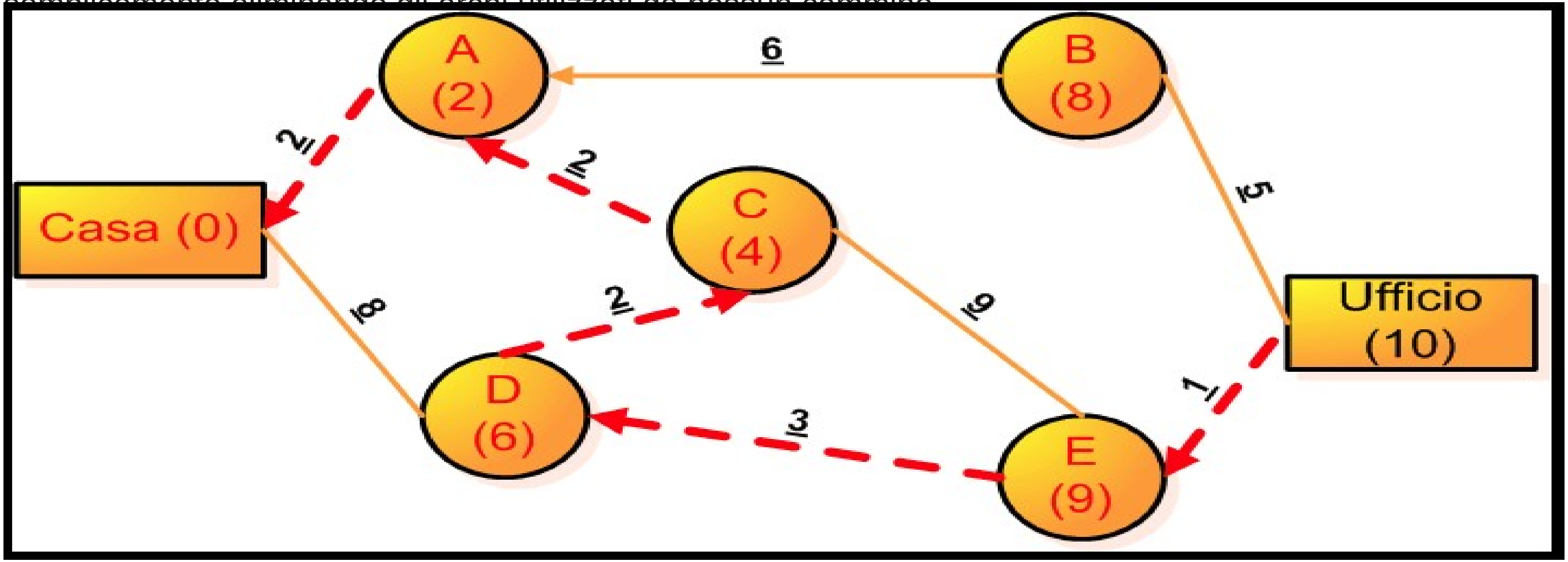
Restano da considerare il nodo E e da aggiornare “ufficio”.



Algoritmo di Dijkstra

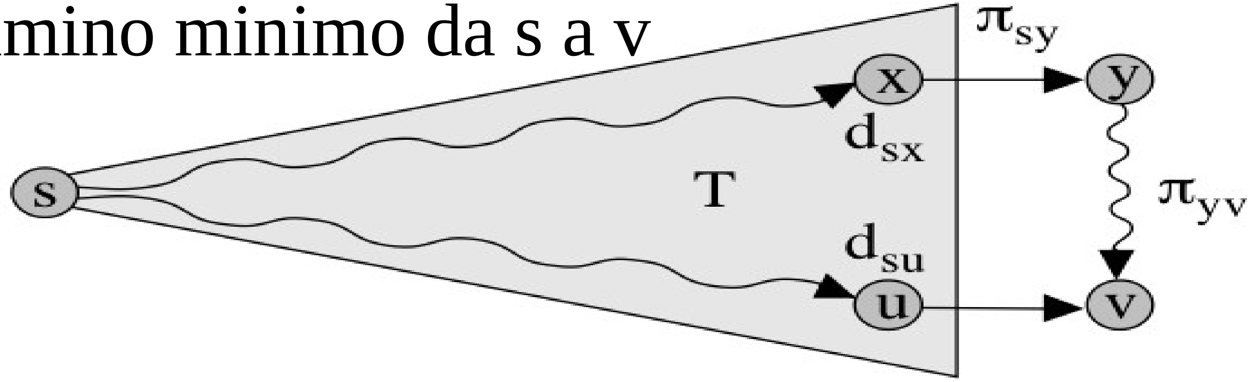
Seguendo all'indietro le frecce si ottiene il percorso minimo da casa a ufficio che misura (come indicato dal potenziale) “10”.

Bisogna notare come questo algoritmo ci dia non solo la distanza minima tra il punto di partenza e quello di arrivo ma la distanza minima di tutti i nodi da quello di partenza, da cui si può realizzare l'albero dei cammini minimi semplicemente eliminando gli archi utilizzati da nessun cammino.



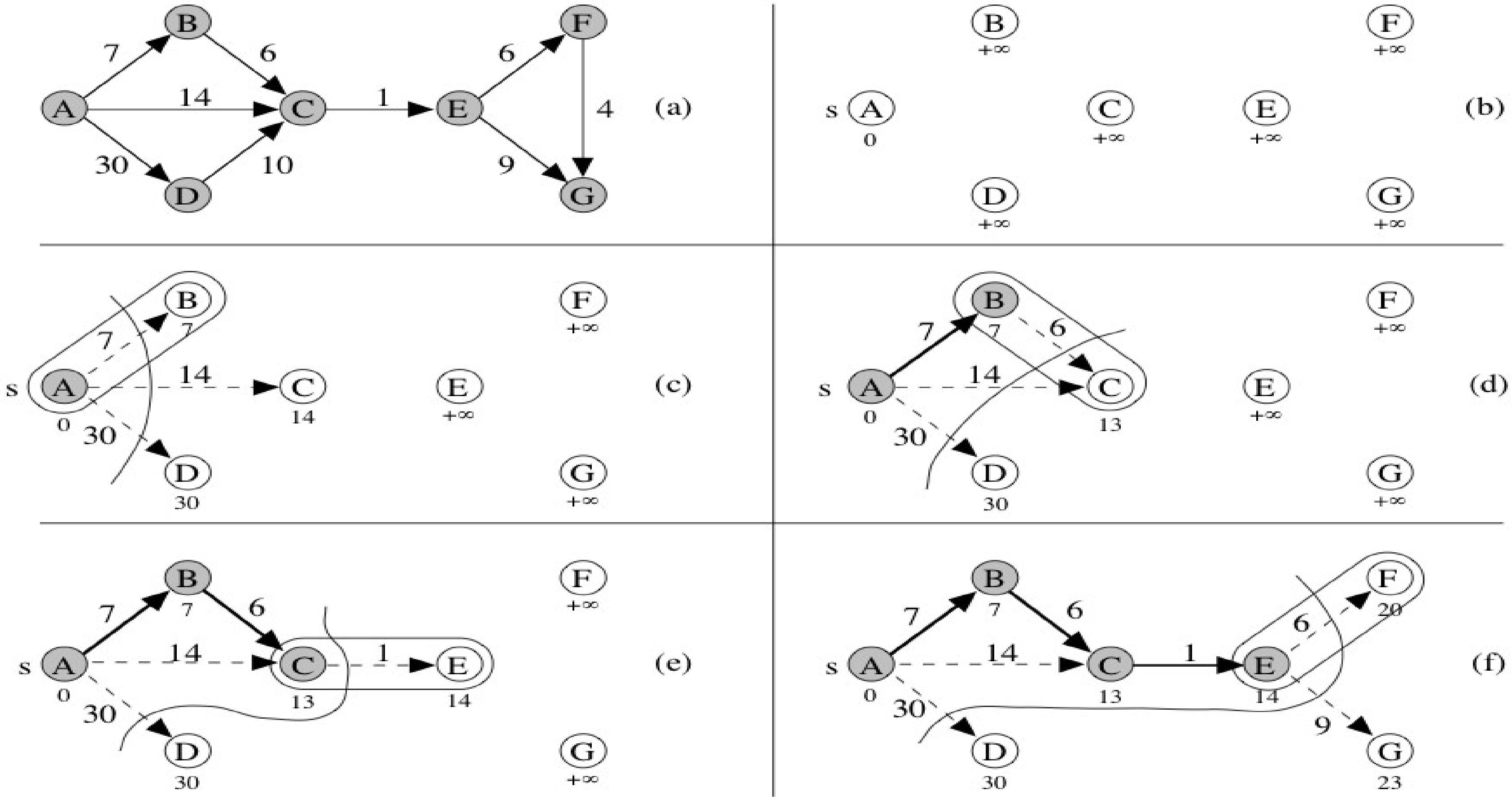
Algoritmo di Dijkstra (ulteriori intuizioni)

Se T è un albero dei cammini minimi radicato in s che non include tutti i vertici raggiungibili da s, l'arco (u,v) tale che u T e v T che minimizza la quantità d_{su}+w(u,v) appartiene a un cammino minimo da s a v

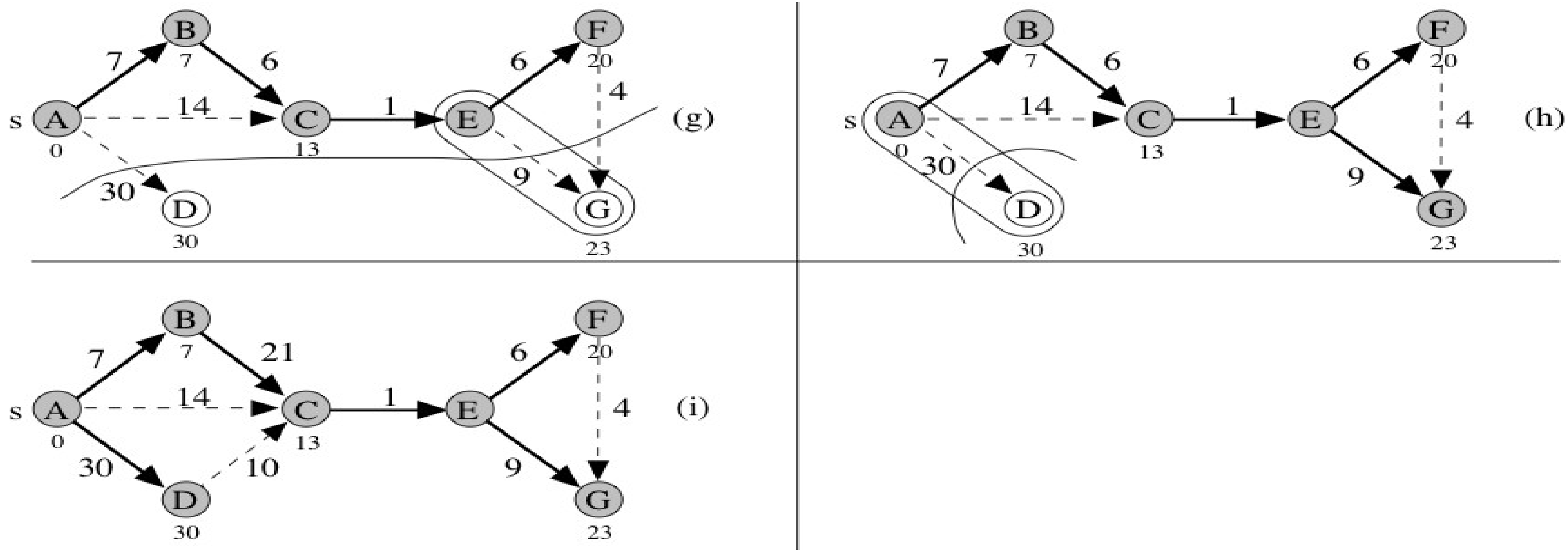


Scegli un arco (u,v) con u T e v T che minimizza la quantità d_{su}+w(u,v), assegna d_{sv} d_{su}+w(u,v), ed aggiungilo a T

Algoritmo di Dijkstra (ulteriori intuizioni)



Algoritmo di Dijkstra (ulteriori intuizioni)



Algoritmo di Dijkstra (ulteriori intuizioni)

Implementiamolo!

Calcolo del minimo albero ricoprente

- Un albero ricoprente (anche detto di copertura, di connessione o di supporto) di un grafo, connesso e con archi non orientati, è un albero che contiene tutti i nodi del grafo e contiene soltanto un sottoinsieme degli archi, cioè solo quelli necessari per connettere tra loro tutti i nodi con uno e un solo cammino.
- Sia dato un grafo G non orientato pesato e supponiamo che esso abbia almeno un albero ricoprente.
- Un minimo albero ricoprente di G è l'albero ricoprente tale che la somma dei pesi dei suoi archi sia minima.

Calcolo del minimo albero ricoprente (Algoritmo di Prim)

Algoritmo goloso:

Scegliamo un nodo qualsiasi, ad esempio il nodo *v*1.

Tra tutti gli adiacenti di *v*1 scegliamo il nodo *v*2 per cui l'arco (*v*1,*v*2) ha peso minimo.

A questo punto scegliamo tra gli adiacenti di *v*1 o *v*2, il nodo *v*3 per cui l'arco (*v*1,*v*3) o (*v*2,*v*3) ha peso minimo.

L'algoritmo goloso così strutturato è detto *algoritmo di Prim*, dal nome dell'informatico che per primo l'ha descritto

L'algoritmo, a parte il fatto che opera su un grafo non orientato, differisce dall'algoritmo di Dijkstra solo nel calcolo della distanza dei nodi, che non è la distanza dal nodo sorgente *s*, ma la distanza minima da un nodo precedentemente scelto, cioè il peso dell'arco utilizzato per connettere il nodo ad un nodo precedentemente visitato. Pertanto, è sufficiente sostituire nell'algoritmo di Dijkstra il calcolo della distanza.