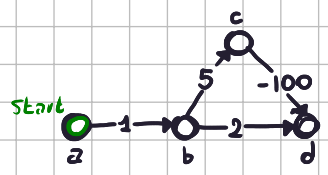
Esercizio 1 (24.4-2, [1]). Sia $G = (V, E)$ un grafo orientato con pesi sugli archi, che possono essere anche negativi ma in cui non sono presenti cicli di peso negativo. Dimostrare che l'algoritmo di Dijkstra applicato su grafi di questo tipo nono calcola necessariamente i cammini di costo minimo tra la sorgente e gli altri vertici del grafo.

Basta considerare il seguente grafo: 

il cammino minimo da a verso d e' $\{a, b, c, d\}$ ma l'algoritmo selezionera' $\{a, b, d\}$ dato che, se $R = \{a, b\}$, si ha che $dist_a(b) + \omega(b, d) < dist_a(b) + \omega(b, c)$. La proposizione sulla quale si basa Dijkstra e' valida se $\omega : E(G) \to \mathbb{R}^+$.

Esercizio 2 (24.3-6, [1]). Sia $G = (V, E)$ un grafo non diretto che rappresenta un network di comunicazioni. Ad ogni arco (u, v) viene associato un valore $r(u, v)$, che è un numero reale con $0 \leq r(u, v) \leq 1$ che rappresenta l'affidabilità dell'arco nel eseguire una comunicazione tra i vertici u e v. Interpretiamo $r(u, v)$ come la probabilità che la trasmissione tra u e v non fallisce e supponiamo che tali probabilità sono indipendenti tra loro. Fornire uno pseudo-codice che dato G e un vertice s restituisce l'albero dei cammini più sicuri da s.

Basta fare una versione modificata di Dijkstra in cui il costo di un cammino e' il prodotto dei pesi.

```
Cammini_sicuri ( G : grafo, r: E(G) → [0,1], s: nodo){
    Dist[n] = {-1, -1, ... -1}
    P[n] = {-1, -1, ... -1}
    R = {s}
    Dist[s] = 1
    P[s] = s
    while ( R ≠ V(G)){
        max_cost = 0
        For each (x, y) ∈ E(G) | x ∈ R ∧ y ∉ R {
            if ( Dist[x] · r(x, y) > max_cost){
                max_cost = Dist[x] · r(x, y)
            }
            Dist[y] = max_cost
            P[y] = x
            R.add(y)
        }
    }
    return P
}
```

Esercizio 3 (15.4-5:6, [1]). Fornire in pseudo-codice un algoritmo che data una sequenza finita di numeri interi X restituisce la lunghezza della più lunga sotto-sequenza strettamente crescente Y. Se, ad esempio, abbiamo che la sequenza X è data da $(1, 3, 8, 5, 4, 2, 6, 0, 1, 2, 8, 9, 5)$ allora si ottiene $Y = (1, 3, 4, 6, 8, 9)$. Implementare questo algoritmo in modo che il tempo di esecuzione sia al più $\mathcal{O}(n^2)$ (ma si può fare anche in $\mathcal{O}(nlog(n))$). Come deve essere modificato l'algoritmo per far si che restituisca una sotto-sequenza strettamente crescente di lunghezza massima?

```
Es3(X:array){
    A[n+1]:array
    A[1]=1
    For(i=2,3...,n){
        m=0
        For(j=i...,1){
            iF(X[j]≤X[i]){
                m=max(T[j],m)
            }
        }
        T[i]=m+1
    }
    Sol:list
    ind
    For(i=n,n-1...,0){
        iF(A[i]
```