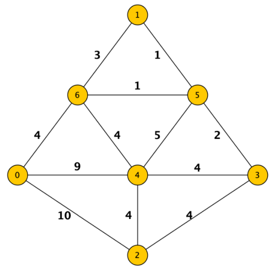
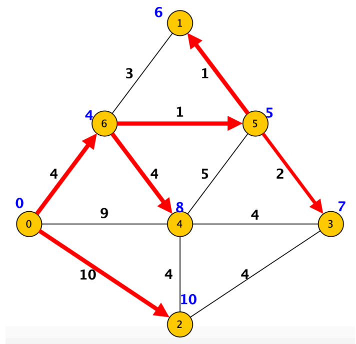
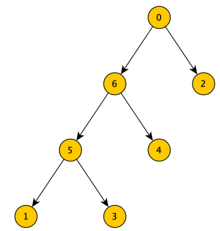
Esercizio 1. Grafi: Applicate l'algoritmo di Dijkstra e determinate la distanza minima di ogni vertice del grafo qui sotto, a partire dal vertice etichettato A. Devono essere rappresentati: il vettore dei genitori, che codifica l'albero dei cammini minimi, il vettore delle distanze minime, e un disegno dell'albero dei cammini minimi.

Nel caso in cui l'algoritmo si trovi a scegliere tra due vertici che hanno la stessa priorità, sceglierà prima quello con l'etichetta che viene prima in ordine alfabetico.



SOLUZIONE esercizio 1:





$$P = \begin{array}{|c|c|c|c|c|c|c|}\hline 0 & 5 & 0 & 5 & 6 & 6 & 0 \\ \hline\end{array} \qquad D = \begin{array}{|c|c|c|c|c|c|c|}\hline 0 & 6 & 10 & 7 & 8 & 5 & 4 \\ \hline\end{array}$$

Esercizio 2. Sottostringa Data una stringa S sull'alfabeto $\{0, 1, 2\}$ vogliamo contare il numero di sottostringhe **012** presenti nella sequenza. Ad esempio:

- per $S = 1210121001$ la risposta deve essere 1 Abbiamo infatti solamente 121**012**1001.
- per $S = 0100120$ la risposta deve essere 4. Abbiamo infatti: **01**001**2**0, **0**100**12**0, 01**0**0**12**0, 010**012**0.

Progettare un algoritmo che risolve il problema in tempo $\Theta(n)$.

Motivare BENE la correttezza e la complessità dell'algoritmo proposto.

SOLUZIONE esercizio 2:

Utilizziamo una matrice T di dimensioni $3 \times n$ dove:

- $T[i,0]$ **conterrà il numero di zeri presenti nei primi i simboli di S**
- $T[i,1]$ **conterrà il numero di sottostringhe 01 presenti nei primi i simboli di S**
- $T[i,2]$ **conterrà il numero di sottostringhe 012 presenti nei primi i simboli di S**

Una volta riempita la matrice T la soluzione sarà nella cella $T[n-1, 2]$.

Resta da definire la funzione ricorsiva che permette di calcolare le varie celle in funzione di quelle gia calcolate.

$$T[i,0] = \begin{cases} 0 & \text{se } i = 0 \text{ e } s[0] \neq 0 \\ 1 & \text{se } i = 0 \text{ e } s[0] = 0 \\ T[i-1,0] & \text{se } i > 0 \text{ e } s[i] \neq 0 \\ T[i-1,0]+1 & \text{se } i > 0 \text{ e } s[i] = 0 \end{cases}$$

$$T[i,1] = \begin{cases} 0 & \text{se } i = 0 \\ T[i-1,1] & \text{se } i > 0 \text{ e } s[i] \neq 1 \\ T[i-1,1]+T[i-1,0] & \text{se } i > 0 \text{ e } s[i] = 1 \end{cases}$$

$$T[i,2] = \begin{cases} 0 & \text{se } i \leq 1 \\ T[i-1,2] & \text{se } i > 1 \text{ e } s[i] \neq 2 \\ T[i-1,2]+T[i-1,1] & \text{se } i > 1 \text{ e } s[i] = 2 \end{cases}$$

Implementazione in Python:

```
def es2(S):
    n=len(S)
    if n<2: return 0
    T=[[0,0,0] for _ in range(n)]
    if S[:2]=='00':
        T[0][0] =1;T[1][0]=2
    elif S[:2]=='01':
        T[0][0] =T[1][0]=1; T[1][1]=1
    elif S[:2]=='10':
        T[0][1]=1
    for i in range (2,n):
        if S[i]=='0':
            T[i][0]=T[i-1][0]+1
            T[i][1]=T[i-1][1]
            T[i][2]=T[i-1][2]
        elif S[i]=='1':
            T[i][0]=T[i-1][0]
            T[i][1]=T[i-1][1]+T[i-1][0]
            T[i][2]=T[i-1][2]
        else:
            T[i][0]=T[i-1][0]
            T[i][1]=T[i-1][1]
            T[i][2]=T[i-1][2]+T[i-1][1]
    return T[n-1][2]
```

Complessità $\Theta(n)$

Esercizio 3. Stampa: Fissiamo n, k e T positivi con $T \leq n\dot{k}$. Definiamo **valida** una sequenza di lunghezza n contenente interi da 0 a k la cui somma è almeno T. Ad esempio se abbiamo $n = 6$, $k = 4$, $T = 11$ allora la sequenza 132312 è **valida** mentre 121213 **non è valida**.

Trovate un algoritmo che dati n ed k stampi tutte e sole le sequenze valide. L'algoritmo deve avere complessità $O(n \cdot k \cdot S(n,k))$ dove $S(n,k)$ è il numero di sequenze valide esistenti.

Motivare BENE la correttezza e la complessità dell'algoritmo proposto, calcolando il numero di nodi interni e di foglie nell'albero di computazione, e mettendoli in corrispondenza con $S(n,k)$.

SOLUZIONE esercizio 3:

- Utilizziamo un algoritmo di backtracking per la stampa di tutte le stringhe lunghe n sull'alfabeto $\{0,\ldots,k\}$ con l'aggiunta di una funzione di taglio.
- Per la funzione di taglio possiamo ragionare come segue: ad ogni inserimento nella sequenza che via via costruiamo teniamo traccia in una variabile *tot* della somma degli interi utilizzati. Dopo l'inserimento dei primi $i-1$ valori nella sequenza, l'inserimento del valore j, con $0 \leq j \leq k$ avverrà se e solo se la sequenza potrà poi completarsi in modo che la somma dei suoi elementi sia almeno T. In altri termini l'inserimento di j avviene se e solo se $tot + j + (n-i-1) * k \geq T$
- grazie alla funzione di taglio la soluzione parziale via via costruita è sempre un prefisso di una soluzione da stampare.

Implementazione:

```
def es3(n,k,T,i=0,tot=0,sol=[]):
    if i==n:
        print(sol)
        return
    for j in range(k+1):
        if tot+j+ (n-i-1)*k>=T:
            sol.append(j)
            es3(n,k,T,i+1,tot+j,sol)
            sol.pop()
```

```
>>> es3(5,1,3)
[0, 0, 1, 1, 1]
[0, 1, 0, 1, 1]
[0, 1, 1, 0, 1]
[0, 1, 1, 1, 0]
[0, 1, 1, 1, 1]
[1, 0, 0, 1, 1]
[1, 0, 1, 0, 1]
[1, 0, 1, 1, 0]
[1, 0, 1, 1, 1]
[1, 1, 0, 0, 1]
[1, 1, 0, 1, 0]
[1, 1, 0, 1, 1]
[1, 1, 1, 0, 0]
[1, 1, 1, 0, 1]
[1, 1, 1, 1, 0]
[1, 1, 1, 1, 1]
```

```
>>> es3(3,5,12)
[2, 5, 5]
[3, 4, 5]
[3, 5, 4]
[3, 5, 5]
[4, 3, 5]
[4, 4, 4]
[4, 4, 5]
[4, 5, 3]
[4, 5, 4]
[4, 5, 5]
[5, 2, 5]
[5, 3, 4]
[5, 3, 5]
[5, 4, 3]
[5, 4, 4]
[5, 4, 5]
[5, 5, 2]
[5, 5, 3]
[5, 5, 4]
[5, 5, 5]
```

- Nota che nell'albero di ricorsione prodotto dall'esecuzione dell'algoritmo un nodo viene generato solo se porta ad una foglia da stampare.
- Possiamo quindi dire che la complessità dell'esecuzione di $es3(n,k,T,i=0,tot=0,sol=[\,])$ richiederà tempo

$$O(S(n) \cdot h \cdot f(n) + S(n,k) \cdot g(n))$$

  dove:

  - $h = n$ è l'altezza dell'albero.
  - $f(n) = \Theta(k)$ è il lavoro di un nodo interno.
  - $g(n) = \Theta(n)$ è il lavoro di una foglia
- Quindi il costo è $O(n \cdot k \cdot S(n,k))$.