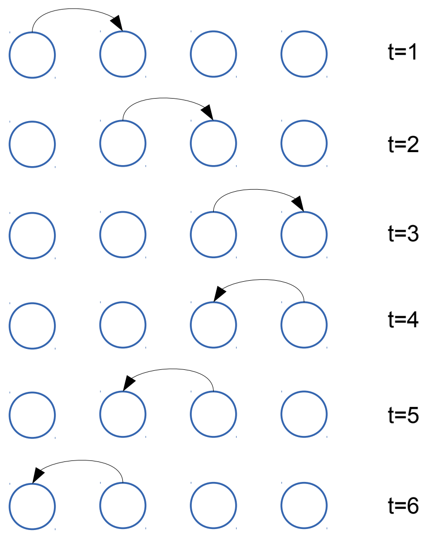
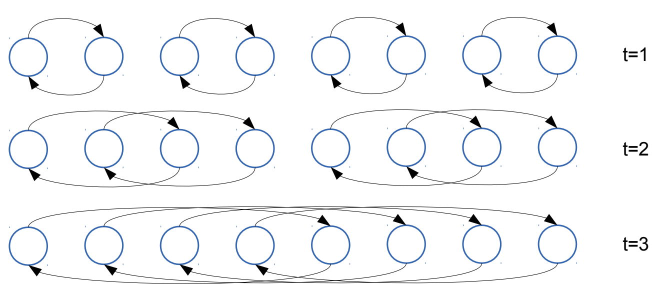
Algoritmi e Strutture Dati
2/09/2013

Esercizio 1

Per quanto riguarda il numero di messaggi, è ovvio che ogni segreto deve essere spedito almeno una volta; quindi sono necessari almeno $\Omega(n)$ messaggi. Un possibile algoritmo è il seguente: al turno i-esimo, la persona i spedisce un messaggio alla persona successiva $i+1$, fino al turno $n-1$; al turno ni, la persona n-esima spedisce un messaggio alla persona precedente $n-1$, la quale spedisce un messaggio alla persona $n-2$, e così via fino al turno $2n-2$ dove la persona 2 spedisce un messaggio alla persona 1, come visualizzato nella figura seguente:



Vengono quindi spediti $2n-2$ messaggi totali in $2n-2$ turni. Ogni messaggio contiene ovviamente tutti i segreti appresi dal mittente fin a quel momento. Questo significa che dal punto di vista del numero di messaggi, $O(n)$ è un limite superiore e quindi l'algoritmo proposto è ottimale.
Sebbene sia possibile ridurre il numero di turni permettendo ai due flussi di messaggi di partire parallelamente, è possibile migliorare l'algoritmo facendo comunicare tutte le persone, come illustrato nella figura seguente:



L'idea è la seguente: dopo il primo turno, tutte le coppie conoscono i segreti di entrambi i membri delle coppie. Dopo il secondo turno, tutte i quartetti conoscono i segreti di tutti i membri dei quartetti, e così via. In altre parole, $O(\log n)$ turni sono sufficienti per comunicare tutti i segreti, al costo aumentato di $O(n \log n)$ messaggi.
Per quanto riguarda un limite inferiore, si consideri un singolo segreto: all'inizio del primo turno, il segreto è noto solo ad un nodo; all'inizio del secondo turno, può essere noto al massimo a due nodi; al terzo turno, può essere noto al massimo quattro nodi; quindi $\Omega(\log n)$ è un limite inferiore al numero di turni, e questo algoritmo è ottimale da questo punto di vista.

Esercizio 2

E' necessario provare le quattro proprietà che definiscono gli alberi Red-Black. Si verifica una volta sola che la radice sia nera, per poi chiamare una procedura ricorsivaa che calcola la profondità nera e verifica che i figli dei nodi rossi siano neri. La proprietà che i nodi **NIL** siano neri si considera automaticamente verificata.

La procedura ricorsiva verifyRBRec(T, h) ritorna una coppia di valori; il primo indica l'altezza nera della foglia più profonda in T, mentre il secondo indica se T rispetta le regole sull'altezza nera e sui figli rossi.

```
boolean verifyRB(TREE T)
    if T.color = RED then
        return false
    (h, b) ← verifyRBRec(TREE T, 0)
    return b
```

```
(integer, boolean) verifyRBRec(TREE T, integer h)
    if T = NIL then return (h, true)
    if T.color = RED and (T.left.color = RED or T.right.color = RED) then
        return (h, false)
    nh ← h + iif(T.color = BLACK, 1, 0)
    (h_L, b_L) ← verifyRBRec(T.left, nh)
    (h_R, b_R) ← verifyRBRec(T.right, nh)
    if not(b_L and b_R) then return (h, false)
    if h_L ≠ h_R then return (h, false)
    return (h_L, true)
```

La complessità è ovviamente $O(n)$.

Esercizio 3

E' possibile semplicemente eseguire una visita in profondità tipo Erdos e poi verificare quale di questi nodi ha distanza minore o uguale a d, con costo $O(m + n)$:

```
integer count(GRAPH G, NODE r, integer d)
    integer count ← 0
    integer[] erdős ← newinteger[1 ... G.n]
    NODE[] p ← newNODE[1 ... G.n]
    erdos(G, r, erdős, p)
    for i ← 1 to G.n do
        if erdős[i] ≤ d then count ← count + 1
    return count
```

Esercizio 4

Utilizziamo la programmazione dinamica per il calcolo e calcoliamo il numero di modi $T[x, i]$ con cui è possibile ottenere un valore x con i primi i dadi:

$$T[x,i] = \begin{cases} \sum_{j=1}^{F[i]} T[x-j, i-1] & i > 0 \wedge x > 0 \\ 1 & x = 0 \wedge i = 0 \\ 0 & \text{altrimenti} \end{cases}$$

L'idea è la seguente: si considera i primi i dadi, e si considerano tutti i possibili valori che possono essere ottenuti dal dado i-esimo: da 1 a $F[i]$, sommando insieme tutti i possibili modi con cui è possibile ottenere il valore restante con un dado in meno.
Il problema di questa versione è che conta tutte le possibili permutazioni; per ovviare a questo, è possibile aggiungere un terzo parametro m che indica il valore minimo del dado che può essere considerato, che deve essere più alto o uguale dei valori già ottenuti:

$$T[x,i,m] = \begin{cases} \sum_{j=m}^{F[i]} T[x-j, i-1, j] & i > 0 \wedge x > 0 \\ 1 & x = 0 \wedge i = 0 \\ 0 & \text{altrimenti} \end{cases}$$

Da questa formulazione è facile ottenere una versione basata su memoization.

```
dice(integer[] F, integer i, integer x, integer m, integer[][][] T)
    if x = 0 and i = 0 then return 1
    if x = 0 or i = 0 then return 0
    if T[x, i, m] = ⊥ then
        T[x, i, m] ← 0
        for j ← m to F[i] do
            T[x, i, m] ← T[x, i, m] + dice(F, i − 1, X − j, j, T)
    return T[x, i, m]
```

Il costo è pari a $O(nXM^2)$, dove M è il dado con il maggior numero di facce. Questo perchè ci sono nXM celle da riempire, ognuna delle quali viene riempita con costo $O(M)$.
Una soluzione alternativa è basata sulla seguente funzione ricorsiva:

$$T[x,i,m] = \begin{cases} T[x-m, i-1, m] + T[x, i, m+1] & i > 0 \wedge x > 0 \wedge m < F[i] \\ T[x-m, i-1, m] & i > 0 \wedge x > 0 \wedge m = F[i] \\ 1 & x = 0 \wedge i = 0 \\ 0 & \text{altrimenti} \end{cases}$$

L'idea è la seguente: come sopra, si considera il dado i-esimo e si cerca di ottenere il valore x, con il terzo parametro m che indica il valore minimo del dado che può essere considerato, che deve essere più alto o uguale dei valori già ottenuti.
Se $m < F[i]$, ci sono ancora dadi e c'è un valore $x > 0$ da ottenere, è possibile scegliere fra: selezionare il valore m per il dado i-esimo, togliendo tale valore da x e considerando quindi cosa succede con $i - 1$ dadi; oppure considerare il dado i innalzando il valore di m. Tali valori vanno sommati. Se $m = F[i]$, il secondo caso non è possibile. I casi base restano uguali.
Una versione memoized di questa equazione ricorsiva conduce ad una complessità di $O(nXM)$, migliore di un fattore M rispetto alla versione precedente.