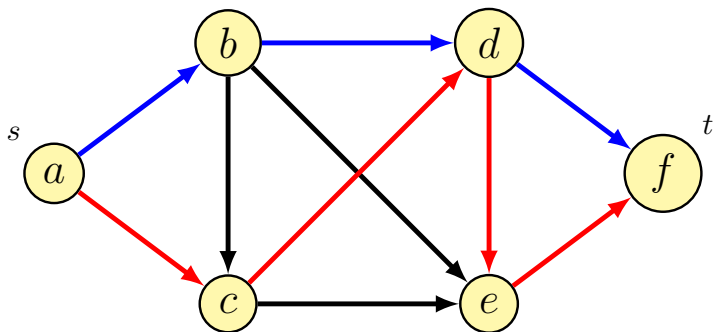


## Cammini indipendenti (Esercizio 1.5 di 15-locale.pdf)

Dato un grafo orientato  $G = (V, E)$  e due vertici  $s, t \in V$ , descrivere un algoritmo che restituisca la dimensione del più grande insieme di cammini **edge-disjoint** fra  $s$  e  $t$ .

Un insieme di cammini si dice **edge-disjoint** se ogni arco del grafo può comparire al massimo in uno dei cammini.



## Torri di controllo (Esercizio 1.7 di 15-locale.pdf)

- Si consideri un insieme di aerei  $A = \{a_1, \dots, a_n\}$  e un insieme di torri di controllo  $T = \{t_1, \dots, t_m\}$ .
- In un certo istante, ogni aereo  $a_i$  si trova alle coordinate  $(a_i.x, a_i.y)$  e ogni torre  $t_j$  si trova alle coordinate  $(t_j.x, t_j.y)$ .
- Vincoli:
  - **Carico**: ogni torre può gestire al più  $L$  aerei;
  - **Distanza**: ogni torre può gestire aerei che si trovino al più a distanza  $d$  da essa.
- Descrivere un algoritmo che assegni ogni aereo ad una torre, rispettando i vincoli sul carico e sulla distanza.
- Discutere correttezza e complessità.

## Prima elementare (Compito 01/02/12)

A mia figlia (prima elementare) è stato chiesto di disegnare tutte le possibili sequenze composte da tre pallini rossi e due pallini gialli.

- 1 Scrivere un algoritmo che stampa tutte le possibili stringhe composte da  $n$  caratteri  $R$  e da  $m$  caratteri  $G$ , per un totale di  $n + m$  caratteri.
- 2 Scrivere un algoritmo che conta tutte queste possibile stringhe – ovviamente senza generarle tutte e poi contandole.
- 3 Calcolare la complessità computazionale degli algoritmi proposti.

## Somma di quadrati (Compito 26/1/16)

Ogni intero positivo  $n$  può essere scritto come somma di quadrati di interi; ad esempio,  $3 = 1^2 + 1^2 + 1^2$ ,  $7 = 2^2 + 1^2 + 1^2 + 1^2$ , mentre  $13 = 3^2 + 2^2$ .

Ovviamente, esistono più modi per esprimere un numero come somma di quadrati; 13 può essere espresso anche come  $2^2 + 2^2 + 2^2 + 1^2$ .

Scrivere un algoritmo che, preso in input  $n$ , restituisce il *numero minimo di quadrati* la cui somma è pari ad  $n$ .

Ad esempio, nel caso di 13,  $3^2 + 2^2$  richiede 2 quadrati, mentre  $2^2 + 2^2 + 2^2 + 1^2$  richiede 4 quadrati, quindi l'algoritmo deve restituire 2.

Discuterne correttezza e complessità.

## Somma di quadrati (Compito 26/1/16)

Scrivere un algoritmo che, dato  $n$ , stampa **tutti** i modi possibili per esprimere  $n$  come somma di quadrati, discutendo correttezza e complessità; ad esempio, con  $n = 13$ , stamperà:

$$1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2$$

$$2^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2$$

$$2^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2$$

$$2^2 + 2^2 + 2^2 + 1^2$$

$$3^2 + 1^2 + 1^2 + 1^2 + 1^2$$

$$3^2 + 2^2$$

Spoiler alert!

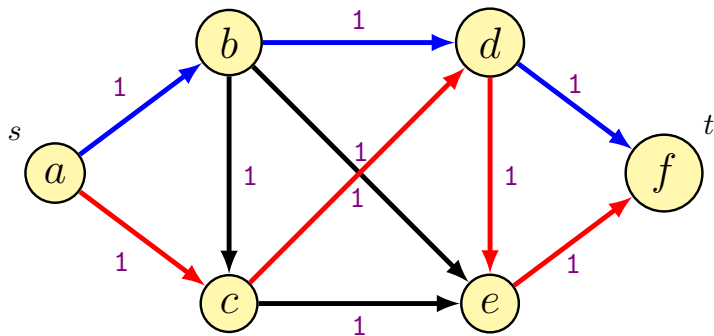
## Cammini indipendenti (Esercizio 1.5 di 15-locale.pdf)

Si costruisce una funzione di capacità  $c$  tale per cui

$$c(x, y) = \begin{cases} 1 & (x, y) \in E \\ 0 & (x, y) \notin E \end{cases}$$

Si consideri il valore del flusso massimo fra  $s$  (sorgente) e  $t$  (pozzo); questo valore corrisponde al numero totale di cammini indipendenti, in quanto essendo la capacità di tutti gli archi pari ad 1, ogni cammino aumentante avrà valore di flusso 1 e i suoi archi avranno capacità residua zero. In questo modo, ogni arco può essere utilizzato in un solo cammino.

## Cammini indipendenti





# Cammini indipendenti

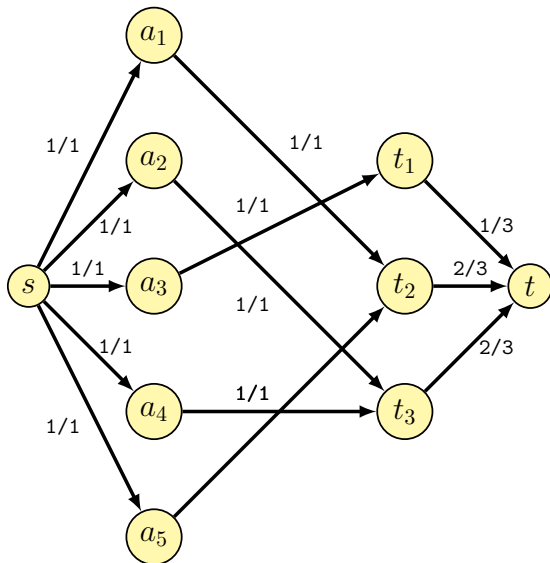
## Complessità

- Seguendo il limite di Ford-Fulkerson, il costo dell'algoritmo è  $O(|f^*|(n + m))$ .
- $|f^*|$  è limitato da  $O(n)$ , perché ci sono al più  $(n - 1)$  archi che escono dalla sorgente (o che entrano nel pozzo)
- Quidi la complessità risultante è  $O(n(m + n)) = O(mn)$ .

## Torri di controllo (Esercizio 1.7 di 15-locale.pdf)

- È possibile associare ad ogni aereo ed ad ogni torre i vertici di un grafo bipartito:
- L'insieme  $V = \{s, t\} \cup \{a_1, \dots, a_n\} \cup \{t_1, \dots, t_m\}$  è costituito dai nodi sorgente, pozzo, i nodi aerei e i nodi torri.
- Nell'insieme  $E$  si inseriscono:
  - un arco  $(s, a_i)$  con capacità 1 fra la sorgente e ogni aereo  $a_i$ ;
  - un arco  $(a_i, t_j)$  con capacità 1 se l'aereo  $a_i$  è entro la portata radio della torre  $t_j$  (ovvero, se  $\sqrt{(a_i.x - t_j.x)^2 + (a_i.y - t_j.y)^2} \leq r$ );
  - un arco  $(t_j, t)$  con capacità  $L$  fra ogni torre e il pozzo  $t$ .

## Torri di controllo (Esercizio 1.7 di 15-locale.pdf)



## Torri di controllo (Esercizio 1.7 di 15-locale.pdf)

- In questo modo:
  - ogni aereo viene assegnato al massimo a una torre di controllo;
  - ogni torre controllo può essere assegnato al massimo a  $L$  aerei.
- Gli archi il cui flusso è pari a 1 corrispondono agli assegnamenti aerei-torri. Il flusso massimo è limitato superiormente da  $n$ .
- Il numero di nodi è pari a  $|V| = n + m + 2$ ;
- il numero di archi è pari a  $|E| = n + m + O(nm)$ .
- Utilizzando il limite di Ford e Fulkerson, il costo dell'algoritmo risultante è quindi  $O(n(|V| + |E|)) = O(n^2m)$ .

## Prima elementare – Backtracking

---

```
printRGRec(char[] S, int i, int n, int m)
```

---

```
if n == 0 and m == 0 then
```

```
    print S
```

```
else
```

```
    if n > 0 then
```

```
        S[i] = "R"
```

```
        printRGRec(S, i + 1, n - 1, m)
```

```
    if m > 0 then
```

```
        S[i] = "G"
```

```
        printRGRec(S, i + 1, n, m - 1)
```

---

---

```
printRG(int n, int m)
```

---

```
int[] S = new char[1 ... n + m]
```

```
printRGRec(S, 1, n, m)
```

---

## Prima elementare – Conteggio inefficiente

---

```
int countRGRec(int n, int m)
```

---

```
if n == 0 or m == 0 then
```

```
    return 1
```

```
else
```

```
    return countRGRec(n - 1, m) + countRGRec(n, m - 1)
```

---

---

```
int countRG(int n, int m)
```

---

```
return countRGRec(n, m)
```

---

Complessità:  $O((n+m)2^{n+m})$  se consideriamo che a ogni passo ho al più due chiamate ricorsive. Più precisamente,  $O\left((n+m)\frac{(n+m)!}{n!m!}\right)$ .

## Prima elementare – Conteggio programmazione dinamica

---

```
int countRG(int  $n$ , int  $m$ )  
  
int[][]  $DP$  = new int[ $0 \dots n$ ][ $0 \dots m$ ]  
for  $i = 0$  to  $n$  do  
     $DP[i][0] = 1$   
for  $j = 0$  to  $m$  do  
     $DP[0][j] = 1$   
for  $i = 1$  to  $n$  do  
    for  $j = 1$  to  $m$  do  
         $DP[i][j] = DP[i - 1][j] + DP[i][j - 1]$   
return  $DP[n][m]$ 
```

---

Complessità:  $O(nm)$

## Prima elementare – Conteggio tramite calcolo combinatorio

---

```
int countRG(int  $n$ , int  $m$ )
```

---

```
return fact( $n + m$ ) / (fact( $n$ ) · fact( $m$ ))
```

---

- Complessità (criterio uniforme):  $O(n + m)$
- Complessità (criterio logaritmico):  $O((n + m)^2 \log^2(n + m))$
- Complessità (criterio logaritmico, fast multiplication, algoritmo efficiente):  $O((n + m)(\log(n + m) \log \log(n + m))^2)$

Peter B. Borwein. *On the complexity of calculating factorials*. In *Journal of Algorithms*, 6(3):376-380, 1985. [PDF]



# Somma di quadrati

Sia  $DP[n]$  il minimo numero di quadrati necessari per esprimere  $n$ .

- Caso base:  $DP[0] = 0$  (non consideriamo il caso  $0^2 = 0$ , in quanto è possibile ottenere 0 non sommando nulla)
- Ricorsione: consideriamo i valori  $t$  tali che  $1 \leq t^2 \leq n$ ; consideriamo quindi i sottoproblemi  $DP[n - t^2]$  e scegliamo fra questi quello che richiede il minor numero di quadrati, aggiungendo 1 per il quadrato considerato.

$$DP[n] = \begin{cases} 0 & n = 0 \\ \min_{1 \leq t \leq \lfloor \sqrt{n} \rfloor} \{DP[n - t^2] + 1\} & n \geq 1 \end{cases}$$

## Somma di quadrati – Programmazione dinamica

---

```
int squareSum(int n)


---


int[] DP = new int[0...n]
DP[0] = 0
for i = 1 to n do
    | DP[i] = +∞
    | for t = 1 to  $\lfloor \sqrt{i} \rfloor$  do
    | | DP[i] = min(DP[i], DP[i - t2] + 1)
    |
return DP[n]
```

---

# Somma di quadrati – Complessità

- Complessità:  $\Theta(n\sqrt{n})$ .
- Lagrange ha dimostrato (**Teorema dei quattro quadrati**) che ogni intero positivo può essere espresso come somma di (al più) quattro quadrati perfetti  
[https://en.wikipedia.org/wiki/Lagrange%27s\\_four-square\\_theorem](https://en.wikipedia.org/wiki/Lagrange%27s_four-square_theorem)
- Michael O. Rabin and Jeffrey Shallit hanno proposto algoritmi probabilistici di complessità polilogaritmica per identificare i quattro quadrati:
  - M. Rabin, J. Shallit. "Randomized Algorithms in Number Theory". Communications on Pure and Applied Mathematics. 39(S1):S239–S256 (1986)
- Un documento più recente è il seguente:
  - P. Pollack, E. Treviño. Finding the four squares in Lagrange's Theorem. [PDF]

## Somma di quadrati – Backtracking

Per risolvere la seconda parte del problema, dobbiamo invece utilizzare la tecnica di backtracking. Una versione semplice per risolvere il problema potrebbe scegliere tutti i possibili valori di  $t$  tali per cui  $1 \leq t^2 \leq n$ , e provare ricorsivamente tutte le possibilità:

---

```
psRec(int n, int[] S, int i)
```

---

```
if n == 0 then
```

```
    print S[1...i - 1]
```

```
else
```

```
    for t = 1 to  $\lfloor \sqrt{n} \rfloor$  do
```

```
        s[i] = t
```

```
        psRec(n - t2, S, i + 1)
```

---

---

```
printSquare(int n)
```

---

```
int[] S = new int[1...n]
```

```
psRec(n, S, 1)
```

---

# Somma di quadrati – Backtracking

- Il vettore  $S$  delle scelte ha dimensione  $n$  (somma di tutti 1).
- La complessità è **superpolinomiale**.
- L'algoritmo appena visto, tuttavia, stampa anche tutte le permutazioni dei quadrati.
- È accettabile per il compito, ma una soluzione migliore evita le permutazioni imponendo un ordine ai valori che vengono sommati.
- Si aggiunge un parametro *limit* per imporre l'ordine
- I valori scelti sono limitati ricorsivamente da *limit*
- Se si è scelto il valore  $t$ , tutte le scelte successive devono essere minori o uguali a quel valore

## Somma di quadrati – Backtracking

---

```
psRec(int n, int[] S, int i, int limit)
```

---

```
if n == 0 then
```

```
    | print S[1 ... i - 1]
```

```
else
```

```
    | for t = 1 to min( $\lfloor \sqrt{n} \rfloor$ , limit) do
```

```
        | s[i] = t
```

```
        | psRec(n - t2, S, i + 1, t)
```

---

```
printSquare(int n)
```

---

```
int[] S = new int[1 ... n]
```

```
psRec(n, S, 1,  $\lfloor \sqrt{n} \rfloor$ )
```

---