L'insegnamento della matematica nell'era digitale (parte 2)

Michele Barbato (Dip. di Informatica "G. Degli Antoni" - UniMI)

email: michele.barbato@unimi.it
web page: https://homes.di.unimi.it/barbato/

Obiettivi

In **questa lezione**:

- **Problemi facili** vs. **difficili**
- **Crittosistemi**
- Problemi su **grafi**
- **Giochi combinatorici**

Problemi facili vs. problemi ardui

- **Misura della difficoltà** di un problema: **tempo** impiegato dal **miglior algoritmo** disponibile per risolverlo
    - Il **tempo** è misurato dal numero di **“passi” eseguiti** dall’algoritmo **su un computer**

Problemi facili vs. problemi ardui

- **Misura della difficoltà** di un problema: **tempo** impiegato dal **miglior algoritmo** disponibile per risolverlo
  - Il **tempo** è misurato dal numero di **"passi" eseguiti** dall'algoritmo **su un computer**

- Normalmente, un algoritmo impiegherà **sempre più passi** al crescere della **taglia delle istanze**
  - Esempio: ci aspettiamo che sia più rapido trovare il **percorso minimo** in una **città piccola** piuttosto che in una **città molto grande**.

Problemi facili vs. problemi ardui

- **Misura della difficoltà** di un problema: **tempo** impiegato dal **miglior algoritmo** disponibile per risolverlo
  - Il **tempo** è misurato dal numero di **"passi" eseguiti** dall'algoritmo **su un computer**

- Normalmente, un algoritmo impiegherà **sempre più passi** al crescere della **taglia delle istanze**
  - Esempio: ci aspettiamo che sia più rapido trovare il **percorso minimo** in una **città piccola** piuttosto che in una **città molto grande**.

- Un **problema** è **"facile"** se ammette un **algoritmo efficiente**, cioè il cui numero di passi **non cresce "troppo"** al crescere della **taglia** delle istanze del problema.
  - Formalmente, il numero di passi deve crescere come un **polinomio di grado costante**

Problemi facili vs. problemi ardui

- **Misura della difficoltà** di un problema: **tempo** impiegato dal **miglior algoritmo** disponibile per risolverlo
  - Il **tempo** è misurato dal numero di **"passi" eseguiti** dall'algoritmo **su un computer**
- Normalmente, un algoritmo impiegherà **sempre più passi** al crescere della **taglia delle istanze**
  - Esempio: ci aspettiamo che sia più rapido trovare il **percorso minimo** in una **città piccola** piuttosto che in una **città molto grande**.
- Un **problema** è **"facile"** se ammette un **algoritmo efficiente**, cioè il cui numero di passi **non cresce "troppo"** al crescere della **taglia** delle istanze del problema.
  - Formalmente, il numero di passi deve crescere come un **polinomio di grado costante**
- Ad oggi, **sembra** che alcuni problemi **non ammettano algoritmi efficienti**

Problemi facili vs. problemi ardui

- **Misura della difficoltà** di un problema: **tempo** impiegato dal **miglior algoritmo** disponibile per risolverlo
  - Il **tempo** è misurato dal numero di **"passi" eseguiti** dall'algoritmo **su un computer**
- Normalmente, un algoritmo impiegherà **sempre più passi** al crescere della **taglia delle istanze**
  - Esempio: ci aspettiamo che sia più rapido trovare il **percorso minimo** in una **città piccola** piuttosto che in una **città molto grande**.
- Un **problema** è **"facile"** se ammette un **algoritmo efficiente**, cioè il cui numero di passi **non cresce "troppo"** al crescere della **taglia** delle istanze del problema.
  - Formalmente, il numero di passi deve crescere come un **polinomio di grado costante**
- Ad oggi, **sembra** che alcuni problemi **non ammettano algoritmi efficienti**
- È ragionevole **supporre** che **questi problemi** siano **"difficili"**
  - Ma è una **congettura tuttora aperta**!

Problemi facili vs. problemi ardui: ostacolo o opportunità?

- Un **problema difficile** è un **ostacolo** quando ci interessa ottimizzare per conseguire uno **scopo**
- Allo stesso tempo la **difficoltà di un problema** può essere **sfruttata** per **mettere al sicuro** delle **informazioni**
    - L'**informazione** viene **rivelata** solo **risolvendo** un **problema difficile**

Crittografie a chiave simmetrica e asimmetrica

- La **crittografia tradizionale** si basa sull'utilizzo di un'**unica chiave** per effettuare **entrambe** le seguenti operazioni:
  - **crittazione** (corrisponde a rendere un messaggio illeggibile, per la trasmissione su un canale insicuro)
  - **decrittazione** (corrisponde a ottenere il messaggio originale a partire da quello trasmesso, illeggibile)
- Questa **crittografia** è quindi detta **simmetrica**
- La crittografia simmetrica permette di cifrare messaggi a un **basso costo computazionale**
- Però **impone la condivisione della chiave** tra mittente e destinatario del messaggio
  - Questo aspetto **espone** i metodi crittografici simmetrici ad **alcune vulnerabilità**

Crittografie a chiave simmetrica e asimmetrica

- Per lo scambio della chiave simmetrica si usa spesso la **crittografia asimmetrica**
- La **crittazione** avviene **con una chiave** K1, la **decrittazione** con un'**altra chiave** K2
- **Requisito**: se il **messaggio** è **cifrato con K1**, allora **non** potrà essere **decifrato con K1**
- Ciò consente di rendere **K1 pubblica**, a patto di tenere **K2 segreta**

Crittografie a chiave simmetrica e asimmetrica

- Per lo scambio della chiave simmetrica si usa spesso la **crittografia asimmetrica**
- La **crittazione** avviene **con una chiave** K1, la **decrittazione** con un'**altra chiave** K2
- **Requisito**: se il **messaggio** è **cifrato con K1**, allora **non** potrà essere **decifrato con K1**
- Ciò consente di rendere **K1 pubblica**, a patto di tenere **K2 segreta**

- K1 consente di trasformare velocemente il messaggio in un "problema" di difficile risoluzione
- K2 consente di risolvere velocemente tale "problema"

Uno zaino speciale

Problema della somma dei sottoinsiemi

- Dato un **insieme A={a[1], a[2], ..., a[n]}** di **interi positivi** e un numero naturale **T,** esiste un **sottoinsieme** di A la cui **somma fa esattamente T**?

Uno zaino speciale

Problema della somma dei sottoinsiemi

- Dato un **insieme A={a[1], a[2], ..., a[n]}** di **interi positivi** e un numero naturale **T**, esiste un **sottoinsieme** di A la cui **somma fa esattamente T**?

Come problema di ottimizzazione:

```
maximize 0x[1]+0x[2]+···+0x[n]
subject to:
    a[1]x[1]+a[2]x[2]+···a[n]x[n] = T
    x[1], x[2],...,x[n] ∈ {0,1}
```

Uno zaino speciale

Problema della somma dei sottoinsiemi

- Dato un **insieme A={a[1], a[2], ..., a[n]}** di **interi positivi** e un numero naturale **T**, esiste un **sottoinsieme** di A la cui **somma fa esattamente T**?

Come problema di ottimizzazione:

```
maximize 0x[1]+0x[2]+···+0x[n]
subject to:
    a[1]x[1]+a[2]x[2]+···a[n]x[n] = T
    x[1], x[2],...,x[n] ∈ {0,1}
```

simile allo zaino già visto

Uno zaino speciale

Problema della somma dei sottoinsiemi

- Dato un **insieme A={a[1], a[2], ..., a[n]}** di **interi positivi** e un numero naturale **T**, esiste un **sottoinsieme** di A la cui **somma fa esattamente T**?

Come problema di ottimizzazione:

```
maximize 0x[1]+0x[2]+···+0x[n]
subject to:
    a[1]x[1]+a[2]x[2]+···a[n]x[n] = T
    x[1], x[2],...,x[n] ∈ {0,1}
```

simile allo zaino già visto

- Problema **"difficile", in generale**
- Problema **"facile" in casi particolari** (e.g., se **A** è **super-crescente**)

Insiemi super-crescenti

- Assumiamo che gli elementi di **A** siano **ordinati in modo crescente**: **a[1] < a[2] < ⋯ < a[n-1] < a[n]**
- **Se a[j] > a[1]+a[2]+⋯a[j-1]** per ogni j=2, …, n l'insieme **A** si dice **super-crescente**

Insiemi super-crescenti

- Assumiamo che gli elementi di **A** siano **ordinati in modo crescente**: **a[n] > a[n-1] > ⋯ > a[2] > a[1]**
- **Se a[j] > a[1]+a[2]+⋯a[j-1]** per ogni j=2, …, n l'insieme **A** si dice **super-crescente**

Esempi:

A = {1, 3, 5, 6, 9} **NON** è super-crescente:

- 3 > 1 ✓

Insiemi super-crescenti

- Assumiamo che gli elementi di **A** siano **ordinati in modo crescente**: **a[n] > a[n-1] > ⋯ > a[2] > a[1]**
- **Se a[j] > a[1]+a[2]+⋯a[j-1]** per ogni j=2, …, n l'insieme **A** si dice **super-crescente**

Esempi:

A = {1, 3, 5, 6, 9} **NON** è super-crescente:

- 3 > 1 ✓
- 5 > 1 + 3 ✓

Insiemi super-crescenti

- Assumiamo che gli elementi di **A** siano **ordinati in modo crescente**: **a[n] > a[n-1] > ⋯ > a[2] > a[1]**
- **Se a[j] > a[1]+a[2]+⋯a[j-1]** per ogni j=2, …, n l'insieme **A** si dice **super-crescente**

Esempi:

A = {1, 3, 5, 6, 9} NON è super-crescente:

- 3 > 1 ✓
- 5 > 1 + 3 ✓
- 6 < 1+3+5 ✗

Insiemi super-crescenti

- Assumiamo che gli elementi di **A** siano **ordinati in modo crescente**: **a[n] > a[n-1] > ⋯ > a[2] > a[1]**
- **Se a[j] > a[1]+a[2]+⋯a[j-1]** per ogni j=2, …, n l'insieme **A** si dice **super-crescente**

Esempi:

A = {1, 3, 5, 11, 23} è super-crescente:

- 3 > 1 ✓
- 5 > 3 + 1 ✓
- 11 > 5 + 3 + 1 ✓
- 23 > 11 + 5+ 3 + 1 ✓

Algoritmo per lo zaino super-crescente

Input: un insieme super-crescente A e un intero positivo T
Output: un sottoinsieme $X \subseteq A$ t.c. $\sum(a \text{ in } X) = T$

1. X := ∅, R:=T
2. Prendi il massimo elemento a* di A tra quelli inferiori a R
3. R ← R-a*
4. A ← A \ {a*}
5. X ← X ∪ {a*}
6. Se R > 0 ricomincia dal passo 2.

Algoritmo per lo zaino super-crescente

Input: un insieme super-crescente A e un intero positivo T
Output: un sottoinsieme X ⊆ A t.c. ∑(a in X) = T

```
1.  X := ∅, R:=T
2.  Prendi il massimo elemento a* di A tra quelli inferiori a R
3.  R ← R-a*
4.  A ← A \ {a*}
5.  X ← X ∪ {a*}
6.  Se R > 0 ricomincia dal passo 2.
```

- In pratica, l'algoritmo precedente fa la **scelta "più ovvia" ad ogni passo**.
- Tali algoritmi si dicono "**algoritmi ghiottoni**" (*greedy algorithms*)

Esercizio

Sia A = {1, 3, 5, 11, 23, 56, 124, 318, 702, 1247}.
Determinare un sottoinsieme $X \subseteq A$ i cui elementi sommino a 888.

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`
- **Prima idea:**
  - scelgo un vettore `A=(a[1], a[2], ...,a[n])` di **n** **interi casuali** e invio il **prodotto scalare** tra **X** e **A**:
    `A·X = x[1]a[1] + x[2]a[2] + ··· + x[n]a[n]`
  - **Esempio di crittazione**:
    se **X** fosse come nell'esempio precedente e scegliessi `A = (10, 12, 45, 56, 61, 89, 100, 149)`, invierei 206.
  - **Processo di decrittazione:**
    Richiede di **risolvere il problema della somma dei sottoinsiemi** definito dall'insieme **A** e **T=206**.
    - È un **problema difficile** (A non è super-crescente) → il messaggio è **ben cifrato**…
    - …anche **troppo** però: il destinatario a sua volta non riuscirà a decrittarlo!

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`
- **Seconda idea:**
  - scelgo un vettore `A=(a[1], a[2], ...,a[n])` di **n** **interi in sequenza super-crescente** e invio il **prodotto scalare** tra **X** e **A**:
    `A·X = x[1]a[1] + x[2]a[2] + ··· + x[n]a[n]`
  - **Esempio di crittazione**:
    se **X** fosse come nell'esempio precedente e scegliessi `A = (10, 12, 45, 68, 61, 150, 341, 679, 1425)`, invierei 1482.
  - **Processo di decrittazione:**
    Richiede di **risolvere il problema della somma dei sottoinsiemi** definito dall'insieme **A** e **T=1482**.
    - È un **problema facile** (A è super-crescente) → stavolta il destinatario **può decrittarlo**…
    - …però **anche gli avversari** in ascolto!

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`

- **Generazione chiave privata**
  - scelgo un vettore `A=(a[1], a[2], ...,a[n])` di **n interi in sequenza super-crescente**
  - scelgo **casualmente** un intero `q > a[1]+ a[2]+...+a[n]`
  - scelgo **casualmente** un intero `r` tale che **MCD(r , q)=1**
  - La **chiave privata** è: `(A, q, r)`

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`

- **Generazione chiave privata**
  - scelgo un vettore `A=(a[1], a[2], ...,a[n])` di **n interi in sequenza super-crescente**
  - scelgo **casualmente** un intero `q > a[1]+ a[2]+...+a[n]`
  - scelgo **casualmente** un intero `r` tale che **MCD(r , q)=1**
  - La **chiave privata** è: `(A, q, r)`

- **Generazione chiave pubblica**
  - Per `i =1, 2, ..., n`, **calcolo** `b[i]` tale che `b[i] = a[i]r (mod q)`
  - Pongo `B=(b[1],b[2],...,b[n])`

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`

- **Generazione chiave privata**
  - scelgo un vettore `A=(a[1], a[2], ...,a[n])` di **n interi in sequenza super-crescente**
  - scelgo **casualmente** un intero `q > a[1]+ a[2]+...+a[n]`
  - scelgo **casualmente** un intero **r** tale che **MCD(r , q)=1**
  - La **chiave privata** è: `(A, q, r)`

- **Generazione chiave pubblica**
  - Per `i =1, 2, ..., n`, **calcolo** `b[i]` tale che `b[i] = a[i]r (mod q)`
  - Pongo `B=(b[1],b[2],...,b[n])`
    - `Esempio: a[1]=6, a[2]=7, a[3]=23, q=37 e r=7.`

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`

- **Generazione chiave privata**
  - scelgo un vettore `A=(a[1], a[2], ...,a[n])` di **n interi in sequenza super-crescente**
  - scelgo **casualmente** un intero `q > a[1]+ a[2]+...+a[n]`
  - scelgo **casualmente** un intero **r** tale che **MCD(r , q)=1**
  - La **chiave privata** è: `(A, q, r)`

- **Generazione chiave pubblica**
  - Per `i =1, 2, ..., n`, **calcolo** `b[i]` tale che `b[i] = a[i]r (mod q)`
  - Pongo `B=(b[1],b[2],...,b[n])`
    - `Esempio: a[1]=6, a[2]=7, a[3]=23, q=37 e r=7. Allora:`
      `b[1]=5 (perché 7x6=37+5)`

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`

- **Generazione chiave privata**
  - scelgo un vettore `A=(a[1], a[2], ...,a[n])` di **n interi in sequenza super-crescente**
  - scelgo **casualmente** un intero `q > a[1]+ a[2]+...+a[n]`
  - scelgo **casualmente** un intero **r** tale che **MCD(r , q)=1**
  - La **chiave privata** è: `(A, q, r)`

- **Generazione chiave pubblica**
  - Per `i =1, 2, ..., n`, **calcolo** `b[i]` tale che `b[i] = a[i]r (mod q)`
  - Pongo `B=(b[1],b[2],...,b[n])`
    - `Esempio: a[1]=6, a[2]=7, a[3]=23, q=37 e r=7. Allora:`
      `b[1]=5 (perché 7x6=37+5), b[2]=12 (perché 7x7=37+12)`

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`

- **Generazione chiave privata**
  - scelgo un vettore `A=(a[1], a[2], ...,a[n])` di **n interi in sequenza super-crescente**
  - scelgo **casualmente** un intero `q > a[1]+ a[2]+...+a[n]`
  - scelgo **casualmente** un intero `r` tale che **MCD(r , q)=1**
  - La **chiave privata** è: `(A, q, r)`

- **Generazione chiave pubblica**
  - Per `i =1, 2, ..., n`, **calcolo** `b[i]` tale che `b[i] = a[i]r (mod q)`
  - Pongo `B=(b[1],b[2],...,b[n])`
    - `Esempio: a[1]=6, a[2]=7, a[3]=23, q=37 e r=7. Allora:`
      `b[1]=5 (perché 7x6=37+5), b[2]=12 (perché 7x7=37+12), b[3]= 13 (perché 23x7=37x4+13)`

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`

- **Generazione chiave privata**
  - scelgo un vettore `A=(a[1], a[2], ...,a[n])` di **n interi in sequenza super-crescente**
  - scelgo **casualmente** un intero `q > a[1]+ a[2]+...+a[n]`
  - scelgo **casualmente** un intero **r** tale che **MCD(r,q)=1**
  - La **chiave privata** è: `(A, q, r)`

- **Generazione chiave pubblica**
  - Per `i =1, 2, ..., n`, **calcolo** `b[i]` tale che `b[i] = a[i]r (mod q)`
  - Pongo `B=(b[1],b[2],...,b[n])`
    - `Esempio: a[1]=6, a[2]=7, a[3]=23, q=37 e r=7. Allora:`
      `b[1]=5 (perché 7x6=37+5), b[2]=12 (perché 7x7=37+12), b[3]= 13 (perché 23x7=37x4+13)`
      Nota: B non è super-crescente, anche se A lo è!

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`

- **Generazione chiave privata**
  - scelgo un vettore `A=(a[1], a[2], ...,a[n])` di **n interi in sequenza super-crescente**
  - scelgo **casualmente** un intero `q > a[1]+ a[2]+...+a[n]`
  - scelgo **casualmente** un intero **r** tale che **MCD(r , q)=1**
  - La **chiave privata** è: `(A, q, r)`

- **Generazione chiave pubblica**
  - Per `i =1, 2, ..., n`, **calcolo** `b[i]` tale che `b[i] = a[i]r (mod q)`
  - Pongo `B=(b[1],b[2],...,b[n])`
  - La **chiave pubblica** è `B=(b[1], b[2], ..., b[n])`

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: `X = (x[1],x[2],...,x[n])`
  Esempio con **n=8** (1 byte): `X = (0,1,1,0,0,0,0,1)`

- **Processo di crittazione**
  - Analogo al precedente: invio il valore `T = X·B` (i.e., **prodotto scalare** del **messaggio** con la **chiave pubblica**)

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: **`X = (x[1],x[2],...,x[n])`**
  Esempio con **n=8** (1 byte): **`X = (0,1,1,0,0,0,0,1)`**

- **Processo di crittazione**
  - Analogo al precedente: invio il valore **`T = X·B`** (i.e., **prodotto scalare** del **messaggio** con la **chiave pubblica**)

- **Processo di decrittazione**
  - Calcolo l'**inverso** **r^{-1} di `r`** modulo **`q`**

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: **`X = (x[1],x[2],...,x[n])`**
  Esempio con **n=8** (1 byte): **`X = (0,1,1,0,0,0,0,1)`**

- **Processo di crittazione**
  - Analogo al precedente: invio il valore **`T = X·B`** (i.e., **prodotto scalare** del **messaggio** con la **chiave pubblica**)

- **Processo di decrittazione**
  - Calcolo l'**inverso** **r^{-1} di r** modulo **q**
    - **Spiegazione:** **r^{-1}** è quel numero **(< q)** t.c. **$rr^{-1} = 1 \pmod q$**
    - Siccome **MCD(r,q)=1**, **esiste sempre** e si **calcola facilmente** (identità di Bézout + algoritmo di Euclide esteso)
    - **Esempio:** **q=31, r=7 → r^{-1}=9** (perché **9x7=1+62=1+31x2**)

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: **`X = (x[1],x[2],...,x[n])`**
  Esempio con **n=8** (1 byte): **`X = (0,1,1,0,0,0,0,1)`**

- **Processo di crittazione**
  - Analogo al precedente: invio il valore **`T = X·B`** (i.e., **prodotto scalare** del **messaggio** con la **chiave pubblica**)

- **Processo di decrittazione**
  - Calcolo l'**inverso** **r^{-1} di r** modulo **q**
  - Calcolo **$t = Tr^{-1} \pmod q$**

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: **`X = (x[1],x[2],...,x[n])`**
  Esempio con **n=8** (1 byte): **`X = (0,1,1,0,0,0,0,1)`**

- **Processo di crittazione**
  - Analogo al precedente: invio il valore **`T = X·B`** (i.e., **prodotto scalare** del **messaggio** con la **chiave pubblica**)

- **Processo di decrittazione**
  - Calcolo l'**inverso** **r^{-1} di `r`** modulo **`q`**
  - Calcolo **`t = Tr`$^{-1}$ `(mod q)`**
  - Risolvo il **problema della somma dei sottoinsiemi** di **`A`** e **`t,`** ossia cerco **`x[1], x[2], ..., x[n]`** t.c.
    - **`a[1]x[1]+a[2]x[2]+···+a[n]x[n] = t`**
    - Siccome **A** è **super-crescente**, posso farlo **facilmente**

Il crittosistema Merkle-Hellman a zaino super-crescente

- **Messaggio da inviare** **n** bit: **`X = (x[1],x[2],...,x[n])`**
  Esempio con **n=8** (1 byte): **`X = (0,1,1,0,0,0,0,1)`**

- **Processo di crittazione**
  - Analogo al precedente: invio il valore **`T = X·B`** (i.e., **prodotto scalare** del **messaggio** con la **chiave pubblica**)

- **Processo di decrittazione**
  - Calcolo l'**inverso** **r^{-1} di `r`** modulo **`q`**
  - Calcolo **`t = Tr`$^{-1}$ `(mod q)`**
  - Risolvo il **problema della somma dei sottoinsiemi** di **`A`** e **`t`**, ossia cerco **`x[1], x[2], ..., x[n]`** t.c.
    - **$a[1]x[1]+a[2]x[2]+\cdots+a[n]x[n] = t$**
    - Siccome **`A`** è **super-crescente**, posso farlo **facilmente**
  - **`X = (x[1],x[2],...,x[n])`** è il **messaggio originale**!

Esercizio guidato

Nota: questo esercizio (guidato) è adattato da un esempio su Wikipedia. Se qualcosa non risulta chiaro, ci si può rifare alla pagina: https://it.wikipedia.org/wiki/Merkle-Hellman

Si consideri il crittosistema Merkle-Hellman basato sulla seguente chiave privata:

- A=(2, 7, 11, 21, 42, 89, 180, 354)
- q=881
- r=588, r^{-1}=442

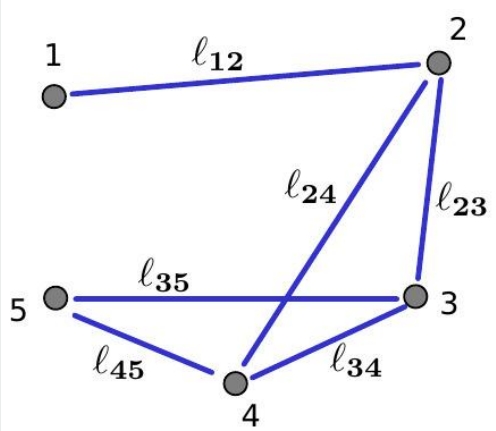
Richieste:

- Si generi una chiave pubblica compatibile per la crittazione e decrittazione di qualunque messaggio in tale crittosistema
- Dato il messaggio X=(0,1,1,0,0,0,0,1), se ne effettui la crittazione con la chiave pubblica generata al punto precedente. Qual è il valore da trasmettere?
- Si decifri in seguito il messaggio cifrato

Grafi

Un **grafo** è uno strumento matematico che permette di **visualizzare rapporti tra oggetti**

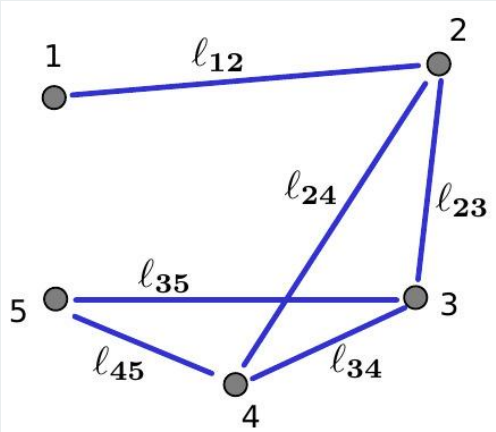
- Gli **oggetti** sono rappresentati da **punti**
- Le **relazioni** sono rappresentate da **linee** che uniscono i punti
- Consideriamo solo **grafi connessi** (hanno un solo "pezzo")



Grafi

Un **grafo** è uno strumento matematico che permette di **visualizzare rapporti tra oggetti**

- Gli **oggetti** sono rappresentati da **punti**
- Le **relazioni** sono rappresentate da **linee** che uniscono i punti
- Consideriamo solo **grafi connessi** (hanno un solo "pezzo")



Terminologia di base:

- **Vertici**: sono i "punti"
- **Lati**: sono le "linee"
- **Grado** di un vertice: numero di lati che tocca il vertice
  - **Esempio**: il vertice 2 ha grado 3
- **Δ**: indica il grado massimo
  - **Esempio**: nel grafo a fianco, **Δ=3**

Problemi su grafi: colorare i lati

Un problema classico su grafi è il seguente:

- **Colorazione dei lati** (*edge-coloring*): si usi il **minimo numero di colori** per colorare **tutti** i lati di un grafo in modo che **due lati** qualsiasi che si **toccano in un vertice** abbiano **colori distinti**

Problemi su grafi: colorare i lati

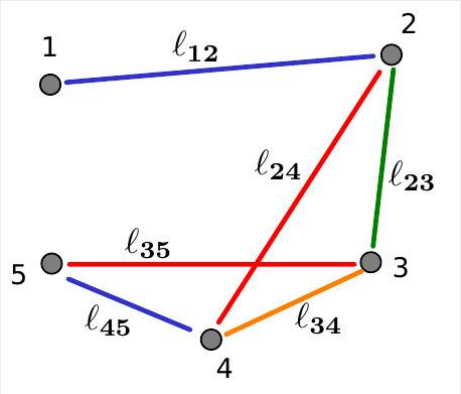
Un problema classico su grafi è il seguente:

- **Colorazione dei lati** (*edge-coloring*): si usi il **minimo numero di colori** per colorare **tutti** i lati di un grafo in modo che **due lati** qualsiasi che si **toccano in un vertice** abbiano **colori distinti**
  - **SOLUZIONE BANALE ma SUB-OTTIMA:** un colore diverso ad ogni lato

Problemi su grafi: colorare i lati

Un problema classico su grafi è il seguente:

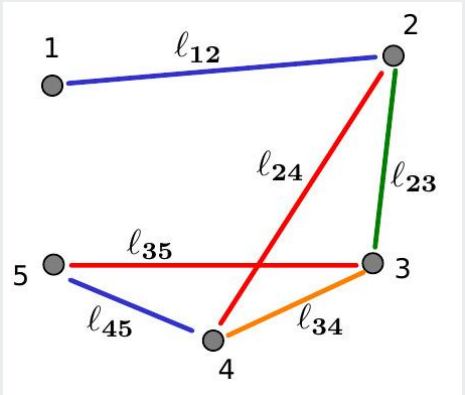
- **Colorazione dei lati** (*edge-coloring*): si usi il **minimo numero di colori** per colorare **tutti** i lati di un grafo in modo che **due lati** qualsiasi che si **toccano in un vertice** abbiano **colori distinti**
  - **Spesso, si può fare meglio!**



Problemi su grafi: colorare i lati

Un problema classico su grafi è il seguente:

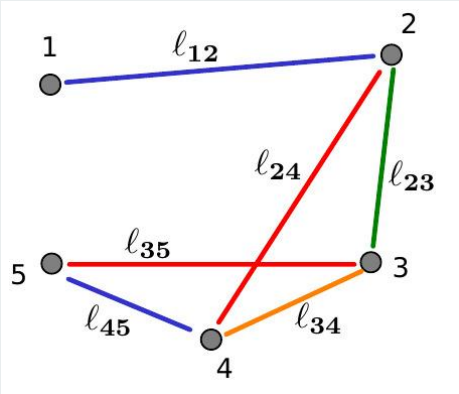
- **Colorazione dei lati** (*edge-coloring*): si usi il **minimo numero di colori** per colorare **tutti** i lati di un grafo in modo che **due lati** qualsiasi che si **toccano in un vertice** abbiano **colori distinti**
- **Risultati di base**:
  - Il **problema** è in generale **arduo** (considerabile difficile teoricamente difficile)
  - Per **ogni grafo**, il **minimo** numero di colori è **compreso tra Δ e Δ+1** (Teorema di Vizing)



Problemi su grafi: colorare i lati

Un problema classico su grafi è il seguente:

- **Colorazione dei lati** (*edge-coloring*): si usi il **minimo numero di colori** per colorare **tutti** i lati di un grafo in modo che **due lati** qualsiasi che si **toccano in un vertice** abbiano **colori distinti**
- **Risultati di base**:
  - Il **problema** è in generale **arduo** (considerabile difficile teoricamente difficile)
  - Per **ogni grafo**, il **minimo** numero di colori è **compreso tra Δ e Δ+1** (Teorema di Vizing)

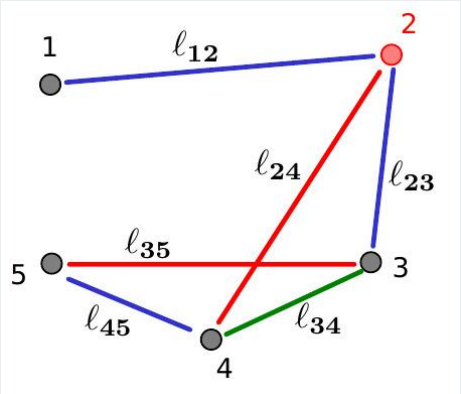


- La soluzione **a lato** usa **4 colori** distinti
- Si può fare **meglio?**

Problemi su grafi: colorare i lati

Un problema classico su grafi è il seguente:

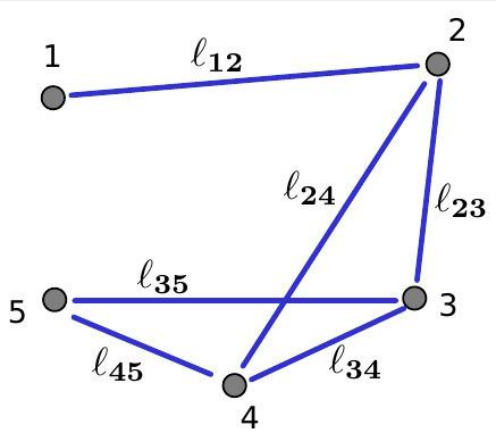
- **Colorazione dei lati** (*edge-coloring*): si usi il **minimo numero di colori** per colorare **tutti** i lati di un grafo in modo che **due lati** qualsiasi che si **toccano in un vertice** abbiano **colori distinti**
- **Risultati di base**:
  - Il **problema** è in generale **arduo** (considerabile difficile teoricamente difficile)
  - Per **ogni grafo**, il **minimo** numero di colori è **compreso tra Δ e Δ+1** (Teorema di Vizing)



- La soluzione **a lato** usa **3 colori** distinti…
- … ma è **scorretta**! (vd. **vertice 2**)

Esercizio

- Impostare un file .mod di AMPL per risolvere il problema di colorare i lati di un grafo *qualsiasi* col minimo numero di colori
- Trovare una colorazione minima per il grafo in basso



Esercizio

Otto squadre devono affrontarsi in un torneo. Il regolamento prevede che ogni squadra affronti una sola volta 5 avversarie diverse, scelte sulla base di un sorteggio. Per la stagione in corso, l'esito del sorteggio è riassunto di seguito:

- La squadra A affronterà B, C, D, E, F
- La squadra B inoltre affronterà D, F, G, H
- La squadra C inoltre affronterà E, F, G, H
- La squadra D inoltre affronterà E, G, H
- La squadra E inoltre affronterà F, G
- La squadra F inoltre affronterà H
- La squadra G inoltre affronterà H

Inoltre, si richiede che ogni squadra affronti al massimo un'avversaria in ogni turno.

- Quanti turni al massimo sono necessari in un calendario che rispetti i requisiti dati sopra?
- Determinare un calendario che usi il numero minimo di turni
- Come cambia la soluzione se si impone che gli incontri A-B, C-E e D-H avvengano nello stesso turno?

Problemi su grafi: colorare i vertici

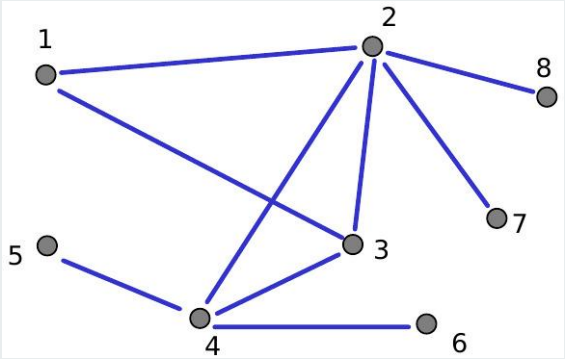
Un altro problema classico su grafi è il seguente:

- **Colorazione dei vertici** (*vertex coloring*): si usi il **minimo numero di colori** per colorare i **vertici** di un grafo in modo che **due vertici adiacenti** (i.e., collegati da un lato) abbiano **colori distinti**.

Problemi su grafi: colorare i vertici

Un altro problema classico su grafi è il seguente:

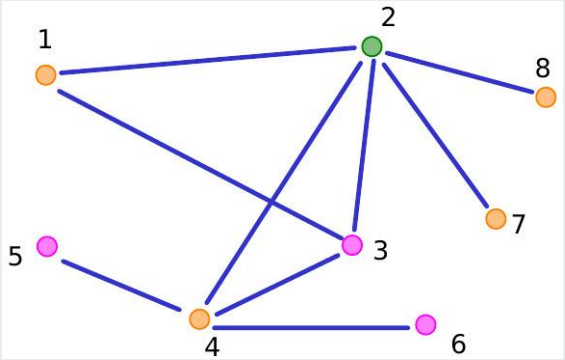
- **Colorazione dei vertici** (*vertex coloring*): si usi il **minimo numero di colori** per colorare i **vertici** di un grafo in modo che **due vertici adiacenti** (i.e., collegati da un lato) abbiano **colori distinti**.



Problemi su grafi: colorare i vertici

Un altro problema classico su grafi è il seguente:

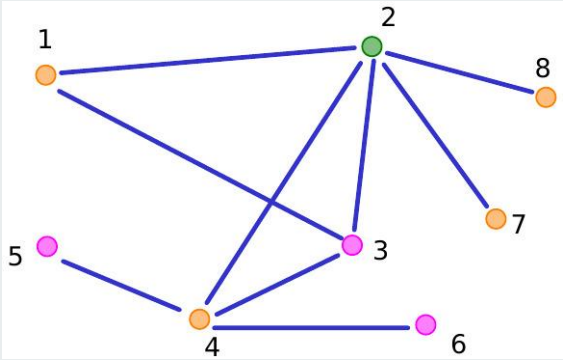
- **Colorazione dei vertici** (*vertex coloring*): si usi il **minimo numero di colori** per colorare i **vertici** di un grafo in modo che **due vertici adiacenti** (i.e., collegati da un lato) abbiano **colori distinti**.



Problemi su grafi: colorare i vertici

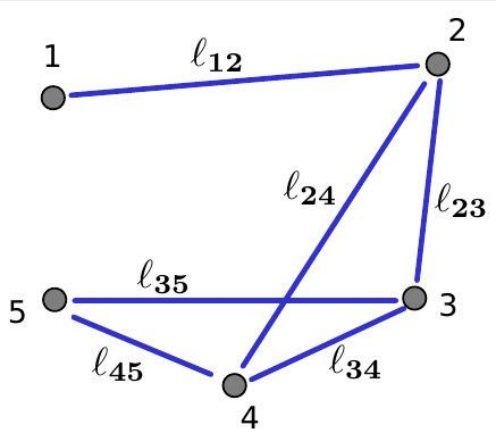
Un altro problema classico su grafi è il seguente:

- **Colorazione dei vertici** (*vertex coloring*): si usi il **minimo numero di colori** per colorare i **vertici** di un grafo in modo che **due vertici adiacenti** (i.e., collegati da un lato) abbiano **colori distinti**.
- **Risultati di base**:
  - Il **problema** è, in generale, **NP-arduo** (considerabile difficile teoricamente difficile)
  - Per **ogni grafo**, il numero **minimo** di colori è al più **Δ+1**…
  - …ma **può essere** anche (molto) **più piccolo** di **Δ**



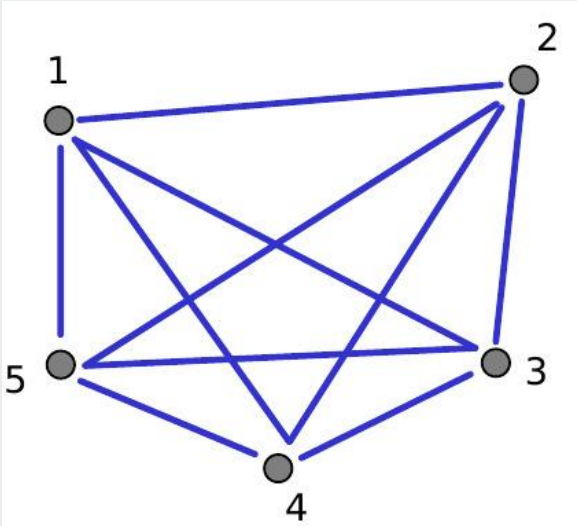
Esercizio

- Impostare un file .mod di AMPL per risolvere il problema di colorare i vertici di un grafo *qualsiasi* col minimo numero di colori
- Trovare una colorazione minima per il grafo in basso



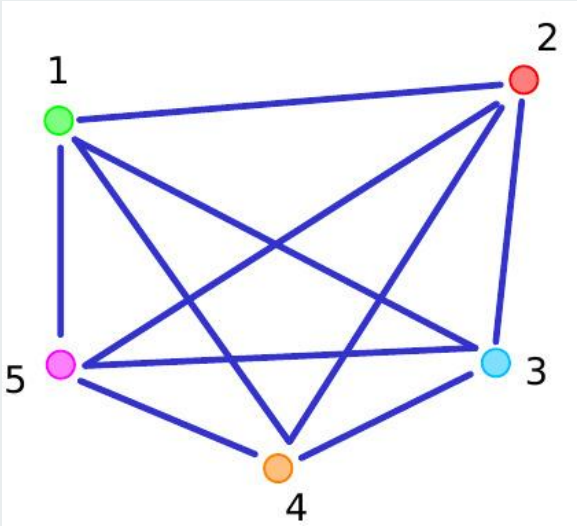
Colorazioni di vertici: cricche, cicli e teorema di Brooks

- Quando un grafo **necessita** di **esattamente Δ+1 colori**?
  - Un esempio è il grafo "**cricca**" (*clique*), in cui **tutti i vertici** sono **mutualmente adiacenti**



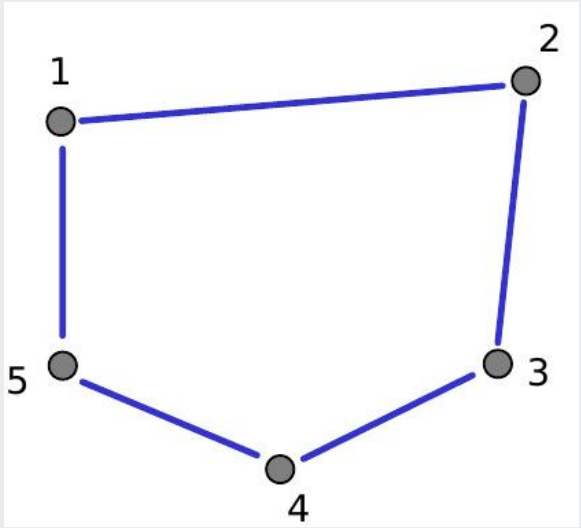
Colorazioni di vertici: cricche, cicli e teorema di Brooks

- Quando un grafo **necessita** di **esattamente Δ+1 colori**?
  - Un esempio è il grafo "**cricca**" (*clique*), in cui **tutti i vertici** sono **mutualmente adiacenti**



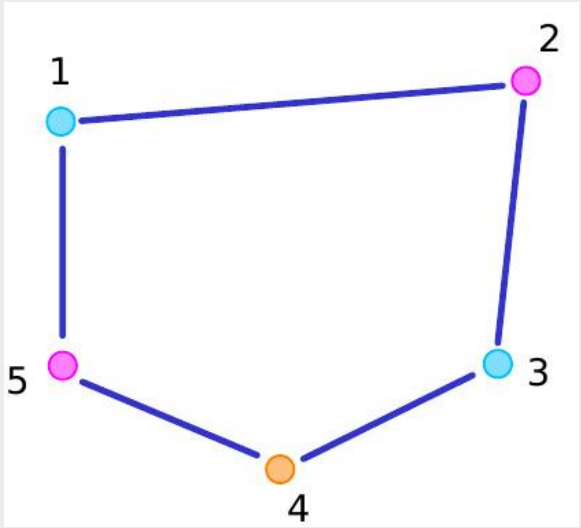
Colorazioni di vertici: cricche, cicli e teorema di Brooks

- Quando un grafo **necessita** di **esattamente Δ+1 colori**?
  - Un esempio è il grafo "**cricca**" (*clique*), in cui **tutti i vertici** sono **mutualmente adiacenti**
  - Un altro esempio è il grafo "**ciclo dispari**", in cui c'è un **numero dispari** di vertici, **tutti** aventi **grado 2**



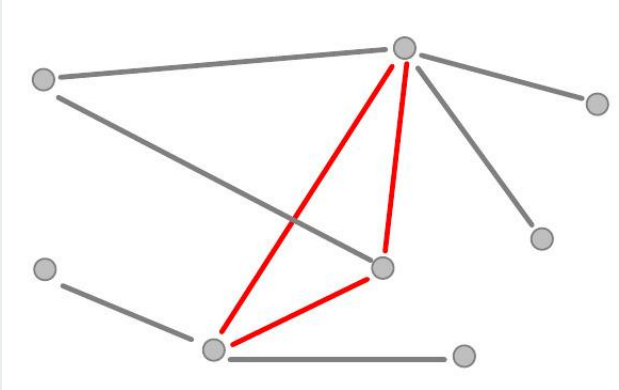
Colorazioni di vertici: cricche, cicli e teorema di Brooks

- Quando un grafo **necessita** di **esattamente Δ+1 colori**?
  - Un esempio è il grafo "**cricca**" (*clique*), in cui **tutti i vertici** sono **mutualmente adiacenti**
  - Un altro esempio è il grafo "**ciclo dispari**", in cui c'è un **numero dispari** di vertici, **tutti** aventi **grado 2**



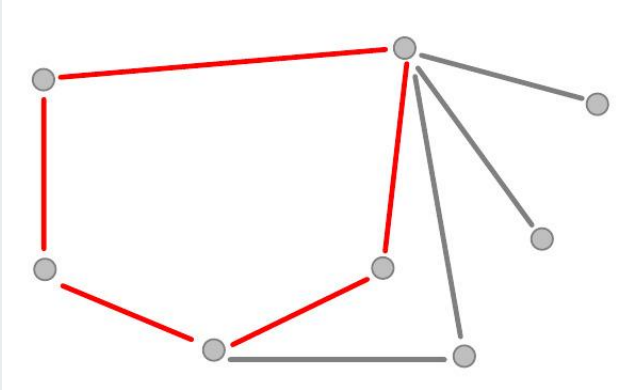
Colorazioni di vertici: cricche, cicli e teorema di Brooks

- Quando un grafo **necessita** di **esattamente Δ+1 colori**?
  - Un esempio è il grafo "**cricca**" (*clique*), in cui **tutti i vertici** sono **mutualmente adiacenti**
  - Un altro esempio è il grafo "**ciclo dispari**", in cui c'è un **numero dispari** di vertici, **tutti** aventi **grado 2**
- Un grafo necessita di **almeno K colori**, dove **K** è il **numero di vertici della sua cricca più grande**



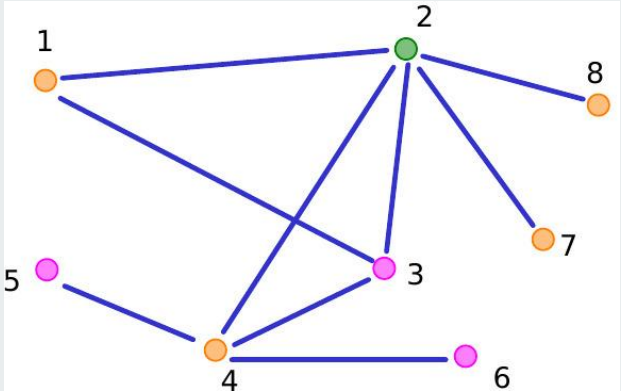
Colorazioni di vertici: cricche, cicli e teorema di Brooks

- Quando un grafo **necessita** di **esattamente Δ+1 colori**?
  - Un esempio è il grafo "**cricca**" (*clique*), in cui **tutti i vertici** sono **mutualmente adiacenti**
  - Un altro esempio è il grafo "**ciclo dispari**", in cui c'è un **numero dispari** di vertici, **tutti** aventi **grado 2**
- Un grafo necessita di **almeno K colori**, dove **K** è il **numero di vertici della sua cricca più grande**
- Un grafo necessita di **almeno 3 colori** qualora **contenga un ciclo dispari**



Colorazioni di vertici: cricche, cicli e teorema di Brooks

- Quando un grafo **necessita** di **esattamente Δ+1 colori**?
  - Un esempio è il grafo "**cricca**" (*clique*), in cui **tutti i vertici** sono **mutualmente adiacenti**
  - Un altro esempio è il grafo "**ciclo dispari**", in cui c'è un **numero dispari** di vertici, **tutti** aventi **grado 2**
- Un grafo necessita di **almeno K colori**, dove **K** è il **numero di vertici della sua cricca più grande**
- Un grafo necessita di **almeno 3 colori** qualora **contenga un ciclo dispari**
- **Teorema di Brooks:** Se un **grafo non** è né una **cricca** né un **ciclo dispari**, allora necessita di **al più Δ colori**



Esercizio

Nel paese di Radiovalle ci sono 12 emittenti radio, che indicheremo, per semplicità, con le lettere A, B, …, K, L.
Ogni ellisse nell'immagine a lato rappresenta l'area in cui una delle emittenti riesce a inviare il proprio segnale.
Per evitare interferenze, occorre assegnare frequenze diverse alle emittenti i cui segnali si sovrappongono in qualche punto.
Per ragioni economiche, si vuole assegnare il numero minimo di frequenze diverse.

Trovare una soluzione ottima senza l'ausilio di AMPL.

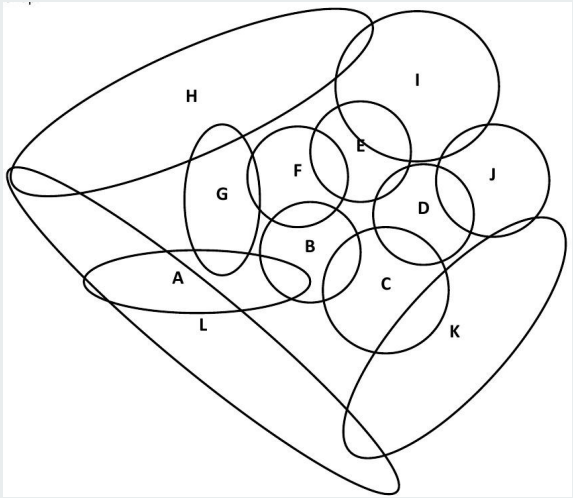
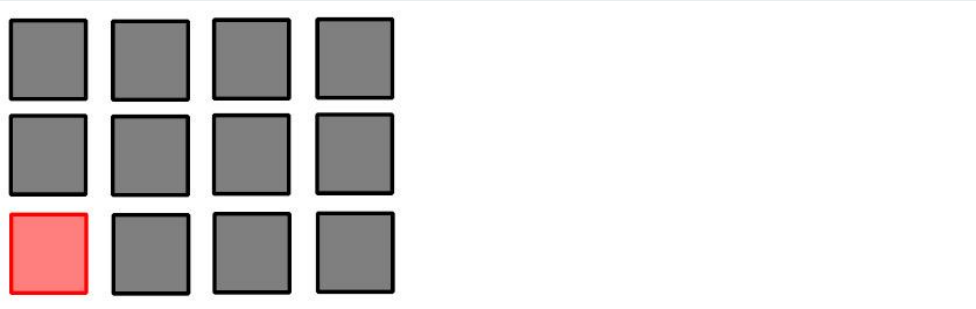


Figura e problema presi da Graph Theory: Puzzles and Games (The Univ. of Edinburgh).
Francesca Iezzi, Ana McKellar, Lukas Cerny, Benedetta Mussati, Patrick Kinnear.

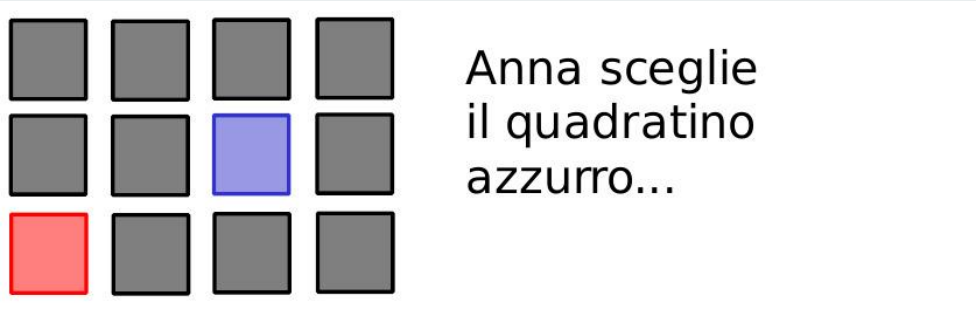
Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**



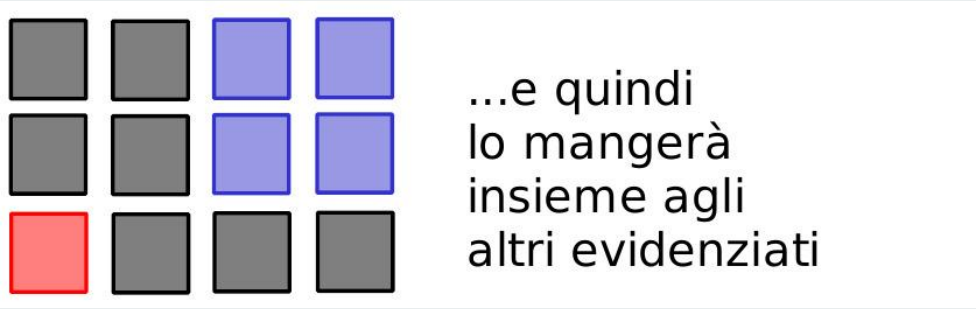
Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**



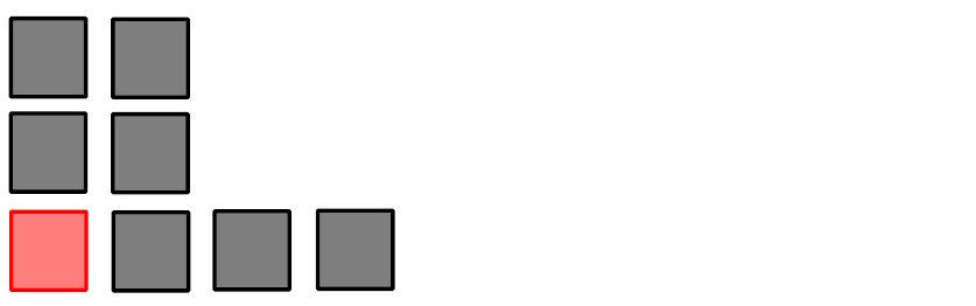
Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**



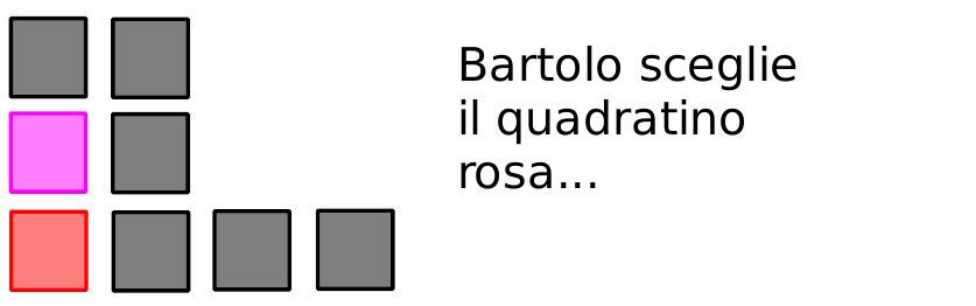
Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**



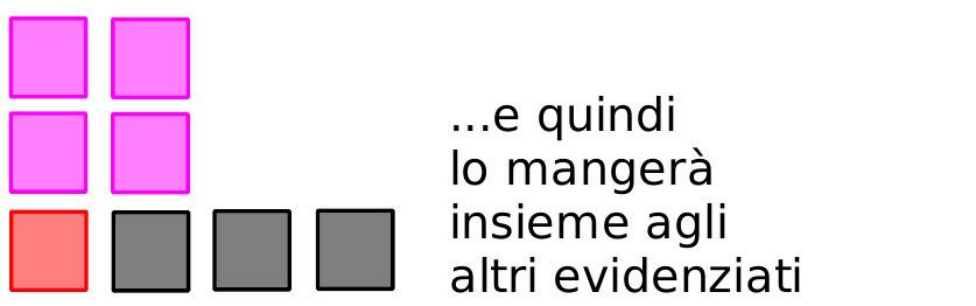
Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**



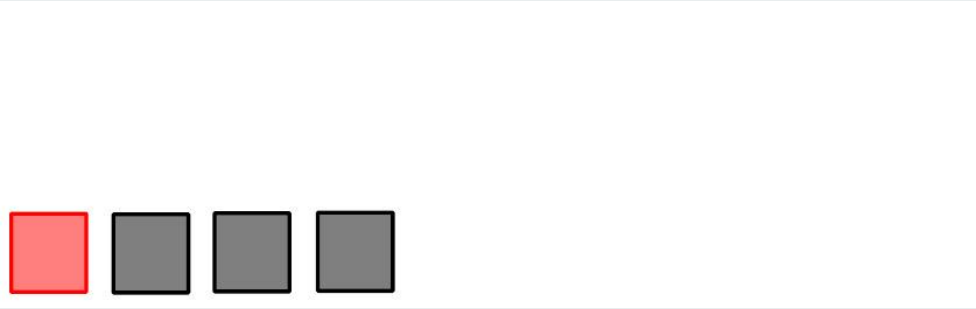
Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**



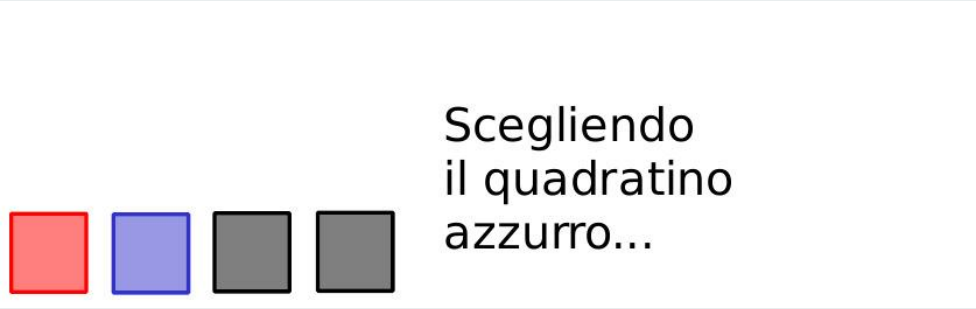
Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**



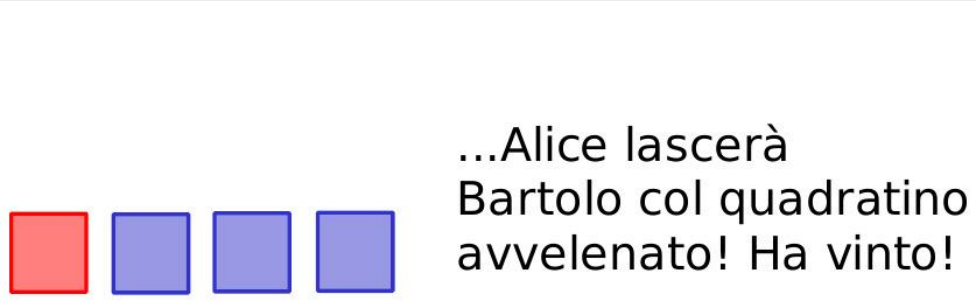
Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**



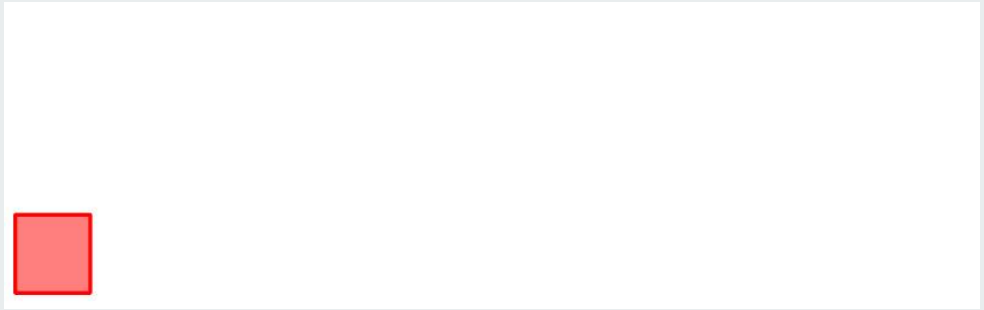
Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**



Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**



Giochi astratti: il “Chomp”

- “Campo di gioco”: tavoletta di cioccolato rettangolare. Il quadrato in basso a sinistra è **avvelenato**
- **Alternandosi**, due giocatori devono **scegliere un quadratino** tra quelli non ancora mangiati, e lo mangiano insieme a **tutti i quadratini** disponibili **alla sua destra e sopra di esso**

Risultati

- Il **primo giocatore** ha sempre una **strategia vincente**…
- …ma è **difficile da determinare**

Per esercitarsi: https://www.math.ucla.edu/~tom/Games/chomp.html

Esercizio

Si consideri il gioco del Chomp su una tavoletta di cioccolata 5x5.

- Qual è una strategia vincente per il giocatore che fa la prima mossa?

Si consideri il gioco del Chomp su una tavoletta di cioccolata quadrata di dimensione arbitraria.

- Qual è una strategia vincente per il giocatore che fa la prima mossa?

Si consideri il gioco del Chomp su una tavoletta 2x6.

- Qual è una strategia vincente per il giocatore che fa la prima mossa?

Si consideri il gioco del Chomp su una tavoletta con due righe e un numero arbitrario di colonne.

- Qual è una strategia vincente per il giocatore che fa la prima mossa?

Giochi astratti: il “Nim”

- “Campo di gioco”: un insieme di pile di oggetti, ognuna con un certo numero di oggetti
- **Alternandosi**, due giocatori **scelgono una sola pila** e rimuovono un **numero arbitrario** di elementi tra quelli rimasti nella pila scelta (da 1 elemento a tutti i possibili elementi). **Vince** il giocatore che prende l’**ultimo oggetto**

Esempio

Pila 1	Pila 2	Pila 3	
3	4	5	(inizio)

Giochi astratti: il “Nim”

- “Campo di gioco”: un insieme di pile di oggetti, ognuna con un certo numero di oggetti
- **Alternandosi**, due giocatori **scelgono una sola pila** e rimuovono un **numero arbitrario** di elementi tra quelli rimasti nella pila scelta (da 1 elemento a tutti i possibili elementi). **Vince** il giocatore che prende l’**ultimo oggetto**

Esempio

Pila 1	Pila 2	Pila 3	
~~3~~ 1	4	5	(Anna prende 2 oggetti dalla pila 1)

Giochi astratti: il “Nim”

- “Campo di gioco”: un insieme di pile di oggetti, ognuna con un certo numero di oggetti
- **Alternandosi**, due giocatori **scelgono una sola pila** e rimuovono un **numero arbitrario** di elementi tra quelli rimasti nella pila scelta (da 1 elemento a tutti i possibili elementi). **Vince** il giocatore che prende l’**ultimo oggetto**

Esempio

Pila 1	Pila 2	Pila 3	
1	4	~~5~~ 2	(Bartolo prende 3 oggetti dalla pila 3)

Giochi astratti: il “Nim”

- “Campo di gioco”: un insieme di pile di oggetti, ognuna con un certo numero di oggetti
- **Alternandosi**, due giocatori **scelgono una sola pila** e rimuovono un **numero arbitrario** di elementi tra quelli rimasti nella pila scelta (da 1 elemento a tutti i possibili elementi). **Vince** il giocatore che prende l’**ultimo oggetto**

Esempio

Pila 1	Pila 2	Pila 3	
1	~~4~~ 0	2	(Anna prende 4 oggetti dalla pila 2)

Giochi astratti: il “Nim”

- “Campo di gioco”: un insieme di pile di oggetti, ognuna con un certo numero di oggetti
- **Alternandosi**, due giocatori **scelgono una sola pila** e rimuovono un **numero arbitrario** di elementi tra quelli rimasti nella pila scelta (da 1 elemento a tutti i possibili elementi). **Vince** il giocatore che prende l’**ultimo oggetto**

Esempio

Pila 1	Pila 2	Pila 3	
1	0	~~2~~ 1	(Bartolo prende 1 oggetto dalla pila 3, e vince)

Giochi astratti: il “Nim”

- “Campo di gioco”: un insieme di pile di oggetti, ognuna con un certo numero di oggetti
- **Alternandosi**, due giocatori **scelgono una sola pila** e rimuovono un **numero arbitrario** di elementi tra quelli rimasti nella pila scelta (da 1 elemento a tutti i possibili elementi). **Vince** il giocatore che prende l’**ultimo oggetto**

Risultati

- Esiste una strategia vincente, facile da calcolare data l’istanza…
- …ma non per forza chi inizia vince

Esercizio

Si consideri la seguente variante del gioco del Nim.
Si ha una sola pila di n oggetti da cui ogni giocatore, durante il proprio turno, può rimuoverne 1, 2 o 3.

- Per quali valori di n il primo giocatore a muovere ha sempre una strategia vincente?
- Indicare una strategia vincente per il primo giocatore in tali casi.

Per ulteriori giochi astratti

Winning ways for your mathematical plays (2 o 4 volumi, a seconda dell'edizione)
di **E.R. Berlekamp, J.H. Conway e R.K. Guy**
(Academic press o AK Peters/CRC Recreational Mathematics Series, a seconda dell'edizione)