Capitolo 5

Cenni su grafi e complessità degli algoritmi

5.1 Grafi

Introduciamo alcune nozioni basilari di teoria dei grafi che saranno utili nei prossimi capitoli.

5.1.1 Grafi non orientati

Un *grafo (non orientato)* G è una coppia ordinata $G = (V, E)$ di insiemi finiti V ed E, dove E è un insieme di coppie non ordinate di elementi distinti di V, cioè $E \subseteq \{\{u, v\} : u, v \in V,\ u \neq v\}$. Gli elementi di V si chiamano *nodi* o *vertici* di G, mentre gli elementi di E si chiamano *spigoli* o *archi* di G.

Per semplicità di notazione, indicheremo uno spigolo $\{u, v\}$ semplicemente con uv. Si noti che, se $uv \in E$, allora $vu = uv$, perché l'ordine con cui consideriamo u e v è irrilevante. Dato un grafo G, denoteremo con $V(G)$ l'insieme dei suoi vertici e con $E(G)$ l'insieme dei suoi spigoli.

Un grafo non orientato viene generalmente raffigurato con $|V|$ punti nel piano, che rappresentano i vertici; ogni spigolo uv è rappresentato tramite un tratto di linea continua congiungente i due punti corrispondenti ai vertici u e v (si veda la parte sinistra della Figura 5.1).

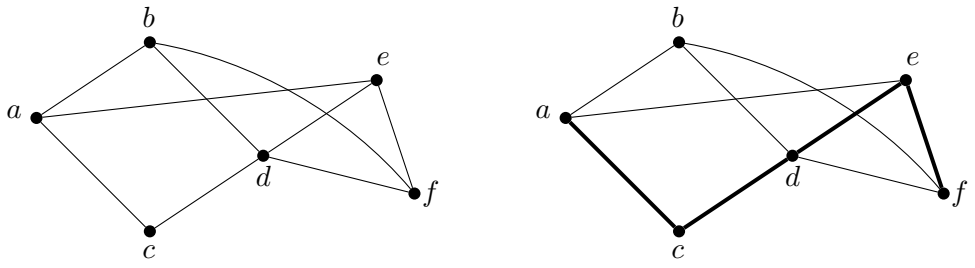

Figura 5.1: Un esempio di grafo non orientato e un cammino di lunghezza 4 tra i nodi a e f.

Dato uno spigolo $e = uv$ di G, i nodi u e v sono detti gli *estremi* di e; inoltre, si dice che u è un *vicino* di v e che lo spigolo e è *incidente* in u (e in v). Due nodi $u, v \in V$ si dicono *adiacenti* se $uv \in E$.

Dato $v \in V$, indichiamo con $\delta(v)$ l'insieme degli spigoli incidenti in v, cioè $\delta(v) := \{uv : uv \in E\}$. Il *grado* di v è il numero naturale $|\delta(v)|$, cioè il numero di spigoli di G incidenti in v, che è anche il numero di nodi adiacenti a v.

Lemma 5.1 *Dato un grafo non orientato $G = (V, E)$, vale la relazione $\sum_{v \in V} |\delta(v)| = 2|E|$.*

Dimostrazione. La sommatoria a primo membro conta, per ogni nodo $v \in V$, quanti spigoli sono incidenti in v; nell'effettuare questo conteggio, ogni spigolo viene considerato due volte (una per ciascuno dei suoi estremi). □

Dato un grafo $G = (V, E)$, un grafo $G' = (V', E')$ è un *sottografo* di G se $V' \subseteq V$ e $E' \subseteq E$. G' è il sottografo di G *indotto* da V' se E' contiene tutti gli spigoli di E con estremi in V', cioè se $E' = \{uv \in E : u, v \in V'\}$; tale sottografo è indicato con la notazione $G[V']$.

Dato un grafo non orientato $G = (V, E)$, un *cammino* P in G è una sequenza alternata di nodi e spigoli

$$v_0, e_1, v_1, e_2, \ldots, v_{k-1}, e_k, v_k, \tag{5.1}$$

dove $k \geq 0$, $v_0, \ldots, v_k$ sono nodi distinti di V ed $e_i = v_{i-1}v_i \in E$ per $i = 1, \ldots, k$. Indicheremo con $V(P)$ l'insieme $\{v_0, \ldots, v_k\}$ dei nodi di P e con $E(P)$ l'insieme $\{e_1, \ldots, e_k\}$ degli spigoli di P. Spesso, per alleggerire la notazione, indicheremo un cammino dando semplicemente la sua sequenza di nodi $v_0, \ldots, v_k$, dato che gli spigoli del cammino sono univocamente determinati dai nodi. I nodi v_0 e v_k si dicono gli *estremi del cammino* P. Un cammino di estremi u e v è un *cammino tra u e v*. La *lunghezza* di un cammino P è il numero di spigoli di P, cioè $|E(P)|$. (Si veda la parte destra della Figura 5.1.) Dato un cammino P come in (5.1), sarà spesso utile indicare con $P_{v_i v_j}$ il *sottocammino* di P che unisce v_i e v_j, dove $0 \leq i \leq j \leq k$. Infine, si dice che u è *connesso* a v se esiste un cammino in G che ha u e v come estremi.

Lemma 5.2 *Dato un grafo non orientato $G = (V, E)$, la relazione "essere connesso a" definita su V è una relazione di equivalenza.*

Dimostrazione. Riflessività e simmetria sono immediate. Per dimostrare che la relazione è transitiva, prendiamo $u, v, w \in V$ e supponiamo che u sia connesso a v e che v sia connesso a w. Allora in G esistono un cammino P tra u e v ed un cammino Q tra v e w. Sia z il primo nodo di Q che si incontra percorrendo P (tale nodo esiste certamente, perché i due cammini hanno in comune almeno un nodo, cioè v). Concatenando i cammini P_{vz} e Q_{zw} otteniamo un cammino tra u e w: infatti tutti gli spigoli di tale cammino appartengono a $E(P) \cup E(P')$ e sono quindi spigoli di G; inoltre, i suoi nodi sono distinti a causa della scelta di z. Dunque u e w sono connessi. □

Dato un grafo $G = (V, E)$, i sottografi indotti dalle classi di equivalenza della relazione "essere connesso a" definita su V sono dette *componenti connesse* di G. In altre parole, un sottografo indotto $G[S]$ (con $\varnothing \neq S \subseteq V$) è una componente connessa di G se e solo se ogni coppia di nodi $u, v \in S$ è connessa in G ed ogni coppia di nodi u, v con $u \in S$ e $v \in V \setminus S$ non è connessa. Il grafo G è detto *connesso* se ogni coppia di nodi $u, v \in V$ è connessa, cioè se G ha una sola componente connessa. Si noti che ogni componente connessa di un grafo qualunque è un sottografo connesso.

Dato un grafo non orientato $G = (V, E)$, un ciclo C in G è una sequenza alternata di nodi e spigoli

$$v_0, e_1, v_1, e_2, \ldots, v_{k-1}, e_k, v_k,$$

dove $k \geq 3$, $v_0, \ldots, v_{k-1}$ sono nodi distinti di V, $e_i = v_{i-1}v_i \in E$ per $i = 1, \ldots, k$ e $v_k = v_0$. Denoteremo con $V(C)$ l'insieme $\{v_0, \ldots, v_{k-1}\}$ dei nodi di C e con $E(C)$ l'insieme $\{e_1, \ldots, e_k\}$ degli spigoli di C. La *lunghezza* di un ciclo C è il numero di spigoli di C, cioè $|E(C)|$. Anche in questo caso, indicheremo spesso un ciclo tramite la sola sequenza di nodi.

Un grafo che non contiene cicli si dice *aciclico*. Si noti che un sottografo di un grafo aciclico è esso stesso aciclico.

Un grafo $G = (V, E)$ è un *albero* se è connesso e aciclico. Un nodo di grado 1 in un albero è chiamato *foglia*.

Lemma 5.3 *Ogni albero con almeno due nodi ha almeno due foglie.*

Dimostrazione. Sia $G = (V, E)$ un albero e si scelga un cammino P in G massimale rispetto all'inclusione, cioè che non sia contenuto in nessun altro cammino di G: lo si può ottenere partendo da un cammino qualunque e aggiungendo spigoli al cammino fin quando possibile. Sia $v_0, v_1 \ldots, v_k$ tale cammino. Poiché G ha almeno due nodi ed è connesso, si ha $k \geq 1$, dunque v_0 e v_k sono nodi distinti. Dimostriamo che v_0 e v_k sono foglie di G. Si consideri un qualunque vertice u adiacente a v_0. Se si avesse $u \notin V(P)$, allora $u, v_0, \ldots, v_k$ sarebbe un cammino in G contenente strettamente P, contro la massimalità di P. Quindi $u \in V(P)$ (e chiaramente $u \neq v_0$, dato che u è un vicino di v_0). Se si avesse anche $u \neq v_1$, allora avremmo $u = v_i$ per qualche $i \in \{2, \ldots, k\}$ e di conseguenza $v_0, v_1, \ldots, v_i, v_0$ sarebbe un ciclo di G, una contraddizione. Quindi l'unica possibilità è $u = v_1$ e di conseguenza $|\delta(v_0)| = 1$, cioè v_0 è una foglia di G. Per simmetria, anche v_k è una foglia di G. □

Teorema 5.4 *Sia $G = (V, E)$ un grafo non orientato. Le seguenti affermazioni sono equivalenti:*

(a) *G è un albero;*

(b) *per ogni $u, v \in V$, esiste esattamente un cammino tra u e v in G;*

(c) *G è aciclico e $|E| = |V| - 1$;*

(d) *G è connesso e $|E| = |V| - 1$.*

Dimostrazione. (a) ⇒ (b) Sia G un albero. Allora G è connesso, quindi esiste un cammino tra ogni coppia di nodi $u, v \in V$. Supponiamo per assurdo che esistano due cammini distinti P e Q tra u e v. Siccome i due cammini sono distinti, esiste un nodo in $V(P) \setminus V(Q)$; ha dunque senso definire w come quel nodo comune a P e Q che precede il primo nodo non comune ai due cammini. Sia z il primo nodo di Q che si incontra percorrendo P a partire da w (un tale nodo esiste certamente). Per la scelta di z, i cammini P_{wz} e Q_{wz} non hanno nodi in comune, eccetto i loro estremi w e z. Inoltre, P_{wz} contiene almeno due spigoli (in caso contrario consisterebbe del solo spigolo wz, cosa non compatibile con la scelta di w e z). Ne segue che questi due cammini insieme formano un ciclo, contro la definizione di albero.

(b) ⇒ (a) Poiché, per ipotesi, esiste un cammino tra ogni coppia di nodi, G è connesso. Supponiamo per assurdo che G contenga un ciclo C dato da $v_0, v_1, \ldots, v_k$, dove $k \geq 3$ (si

tenga presente che $v_k = v_0$). È semplice verificare che allora $v_0, v_1, \ldots, v_{k-1}$ e v_{k-1}, v_k sono due cammini diversi tra v_0 e v_{k-1}, una contraddizione.

(a) $\Rightarrow$ (c), (d) È sufficiente dimostrare che se G è un albero allora $|E| = |V| - 1$. La dimostrazione è per induzione su $|V|$. L'affermazione è chiaramente vera se $|V| = 1$, in quanto in tal caso $|E| = 0$. Supponiamo ora $|V| \geq 2$. Per il Lemma 5.3, G ha una foglia v; sia u l'unico nodo adiacente a v in G. Si consideri il sottografo $G' = (V', E')$ indotto da $V' = V \setminus \{v\}$ e si osservi che $E' = E \setminus \{uv\}$. G' è aciclico perché tale è G. Verifichiamo ora che G' è connesso. Dati due nodi in $V \setminus \{v\}$, esiste un cammino P in G che li connette; allora $V(P)$ non contiene il nodo v, perché ogni nodo che fa parte di un cammino ma non ne è un estremo ha grado almeno 2, mentre v è un foglia; quindi, P è anche un cammino in G', da cui segue che G' è connesso. Dunque G' è un albero. Possiamo allora applicare l'ipotesi induttiva e concludere che $|E'| = |V'| - 1$. Poiché $|E| = |E'| + 1$ e $|V| = |V'| + 1$, otteniamo $|E| = |V| - 1$.

(c) $\Rightarrow$ (a) Sia k il numero di componenti connesse del grafo G, che indichiamo con $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$. Per $i = 1, \ldots, k$, il grafo G_i è connesso (per definizione di componente connessa) ed aciclico (perché sottografo di un grafo aciclico), dunque è un albero. Poiché abbiamo già dimostrato "(a) $\Rightarrow$ (c)", abbiamo $|E_i| = |V_i| - 1$ per $i = 1, \ldots, k$. Per definizione di componente connessa, gli insiemi $V_1, \ldots, V_k$ formano una partizione di V, dunque $\sum_{i=1}^{k} |V_i| = |V|$. Inoltre, ogni spigolo di E appartiene ad uno ed uno solo degli insiemi $E_1, \ldots, E_k$, quindi $|E| = \sum_{i=1}^{k} |E_i|$. Otteniamo allora $|E| = \sum_{i=1}^{k} |E_i| = \sum_{i=1}^{k} (|V_i| - 1) = |V| - k$. Poiché, per ipotesi, $|E| = |V| - 1$, deve valere $k = 1$, cioè G è connesso.

(d) $\Rightarrow$ (a) La dimostrazione è per induzione su $|V|$. L'affermazione è chiaramente vera se $|V| = 1$. Sia ora $|V| \geq 2$. Dimostriamo prima di tutto che G contiene un nodo di grado 1. Infatti, se si avesse $|\delta(v)| \geq 2$ per ogni $v \in V$, dal Lemma 5.1 ricaveremmo $|E| \geq |V|$, una contraddizione. Pertanto deve esistere un nodo $v \in V$ tale che $|\delta(v)| \leq 1$; ma G è connesso, per cui v non può avere grado zero. Dunque esiste un nodo v di grado 1. Ripetendo gli argomenti visti nella dimostrazione di "(a) $\Rightarrow$ (c), (d)", deduciamo che il sottografo $G' = (V', E')$ indotto da $V' = V \setminus \{v\}$ è connesso e $|E'| = |V'| - 1$. Per induzione, G' è dunque aciclico. Siccome nessun ciclo di G può avere v tra i suoi vertici (perchè ogni nodo di un ciclo ha grado almeno 2), concludiamo che anche G è aciclico. □

Dalla dimostrazione del teorema precedente, è immediato osservare che le condizioni (c) e (d) sono equivalenti alle seguenti, apparentemente più deboli:

(c′) G è aciclico e $|E| \geq |V| - 1$;

(d′) G è connesso e $|E| \leq |V| - 1$.

5.1.2 Grafi orientati

Un *grafo orientato* (o *digrafo*) D è una coppia ordinata $D = (V, A)$ di insiemi finiti V ed A, dove A è un insieme di coppie ordinate di elementi distinti di V, cioè $A \subseteq \{(u, v) \in V \times V : u \neq v\}$. Gli elementi di V si chiamano *nodi* o *vertici* di D, mentre gli elementi di A si chiamano *archi* di D.

Un grafo orientato viene generalmente raffigurato con $|V|$ punti nel piano, che rappresentano i vertici; un arco (u, v) è raffigurato tramite una linea continua orientata da u verso v (si veda la parte sinistra della Figura 5.2).

Dato un arco $e = (u, v)$ di un grafo orientato, u è detto la *coda* di e e v la *testa* di e. Diremo anche che l'arco e *esce* da u ed *entra* in v. Si noti che $(u, v) \neq (v, u)$. Per ogni nodo

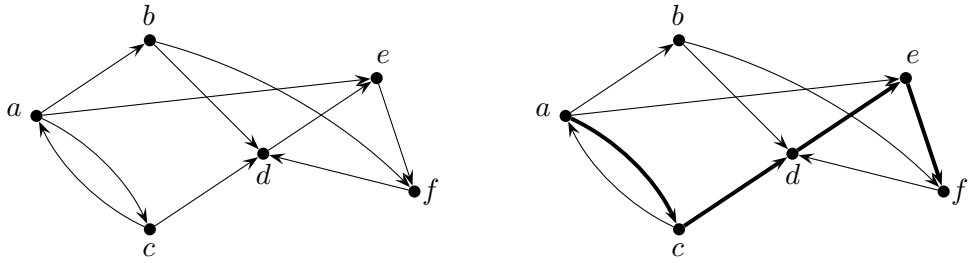

Figura 5.2: Un esempio di grafo orientato e un cammino orientato di lunghezza 4 da a a f.

$v \in V$, indicheremo con $\delta^+(v)$ l'insieme degli archi che escono da v e con $\delta^-(v)$ l'insieme degli archi che entrano in v. Si verifica facilmente che per ogni grafo orientato $D = (V, A)$ vale

$$|A| = \sum_{v \in V} |\delta^-(v)| = \sum_{v \in V} |\delta^+(v)|. \tag{5.2}$$

Dato un grafo orientato $D = (V, A)$, un *cammino orientato* P in D è una sequenza alternata di nodi e archi $v_0, e_1, v_1, e_2 \ldots, v_{k-1}, e_k, v_k$, dove $k \geq 0$, $v_0, \ldots, v_k$ sono nodi distinti di V e $e_i = (v_{i-1}, v_i) \in A$ per $i = 1, \ldots, k-1$. Indicheremo con $V(P)$ l'insieme $\{v_0, \ldots, v_k\}$ dei nodi di P e con $A(P)$ l'insieme $\{e_1, \ldots, e_k\}$ degli archi di P. Il cammino P è detto un cammino dal nodo v_0 al nodo v_k. Come nel caso non orientato, spesso indicheremo solo la sequenza di nodi di un cammino. La *lunghezza* di P è il numero di archi di P, cioè $|A(P)|$. (Si veda la parte destra della Figura 5.2.)

In modo del tutto analogo al Lemma 5.2 si dimostra la seguente proprietà.

Lemma 5.5 *Dato un grafo orientato $D = (V, A)$, la relazione "esiste un cammino da u a v in D" definita su V è riflessiva e transitiva.*

Si noti che la relazione del lemma precedente in generale non è simmetrica.

Dato un grafo orientato $D = (V, A)$, un *ciclo orientato* C in D è una sequenza alternata di nodi e archi $v_0, e_1, v_1, e_2 \ldots, v_{k-1}, e_k, v_k$, dove $k \geq 2$, $v_0, \ldots, v_{k-1}$ sono nodi distinti di V, $e_i = (v_{i-1}, v_i) \in A$ per $i = 1, \ldots, k$ e $v_0 = v_k$. Indicheremo con $V(C)$ l'insieme $\{v_0, \ldots, v_{k-1}\}$ dei nodi di C e con $A(C)$ l'insieme $\{e_1, \ldots, e_k\}$ degli archi di C. Al solito, gli archi saranno spesso lasciati sottintesi. La *lunghezza* di un ciclo C è il numero di archi di C, cioè $|A(C)|$.

Dato un grafo orientato $D = (V, A)$, il grafo non orientato $G = (V, E)$ definito da $E = \{uv : (u, v) \in A \text{ oppure } (v, u) \in A\}$ è detto il grafo non orientato *sottostante* a D. In altri termini, G è il grafo non orientato ottenuto da D ignorando l'orientamento degli archi e mantenendo una sola copia di ogni spigolo così ottenuto. Ad esempio, il grafo in Figura 5.1 è il grafo non orientato sottostante al grafo orientato in Figura 5.2. Si noti che vale sempre la relazione $|E| \leq |A|$ e che si ha $|E| = |A|$ se e solo se D non contiene coppie di archi tra loro opposti, cioè coppie di archi del tipo (u, v) e (v, u).

Un *cammino non orientato* in D è una sequenza di nodi e archi di D che definisce un cammino nel grafo non orientato sottostante a D. Ad esempio, in Figura 5.2, la sequenza b, d, f definisce un cammino non orientato di D, ma non un cammino orientato. La definizione di *ciclo non orientato* può essere data in modo analogo.

Un grafo orientato $D = (V, A)$ si dice *connesso* se il suo grafo sottostante è connesso, mentre si dice *fortemente connesso* se, per ogni coppia di nodi $u, v \in V$, esistono cammini

orientati sia da u a v che da v ad u. Si noti che il grafo orientato in Figura 5.2 è connesso ma non fortemente connesso, in quanto non esiste alcun cammino orientato dal nodo b al nodo a.

Dati un grafo orientato $D = (V, A)$ ed un nodo $r \in V$, D è un'*arborescenza con radice* r se $|A| = |V| - 1$ e per ogni $v \in V$ esiste un cammino orientato da r a v.

Lemma 5.6 *Sia $D = (V, A)$ un grafo orientato e sia $r \in V$. Allora D è un'arborescenza con radice r se e solo se il grafo sottostante a D è un albero e valgono le proprietà $|\delta^-(r)| = 0$ e $|\delta^-(v)| = 1$ per ogni $v \in V \setminus \{r\}$.*

Dimostrazione. Sia $G = (V, E)$ il grafo sottostante a D. Supponiamo che G sia un albero, $|\delta^-(r)| = 0$ e $|\delta^-(v)| = 1$ per ogni $v \in V \setminus \{r\}$. Per la relazione (5.2),

$$|A| = \sum_{v \in V} |\delta^-(v)| = |\delta^-(r)| + \sum_{v \in V \setminus \{r\}} |\delta^-(v)| = |V| - 1,$$

dunque la condizione $|A| = |V| - 1$ è soddisfatta. Per concludere che D è un'arborescenza con radice r, resta da verificare che per ogni $v_0 \in V$ esiste un cammino orientato da r a v_0. Questo è chiaramente vero se $r = v_0$. Sia quindi $r \neq v_0$. Per ipotesi, $|\delta^-(v_0)| = 1$, cioè esiste un nodo $v_1 \in V$ tale che $(v_1, v_0) \in A$. Se $v_1 = r$, allora v_1, v_0 è il cammino cercato; altrimenti $|\delta^-(v_1)| = 1$ e dunque esiste $v_2 \in V$ tale che $(v_2, v_1) \in A$. Supponiamo $v_2 = v_0$. Allora D contiene gli archi (v_0, v_1) e (v_1, v_0), che sono tra loro opposti. Come osservato più sopra, sotto questa condizione si ha $|E| < |A|$, da cui $|E| < |V| - 1$, contro l'ipotesi che G sia un albero (si veda la condizione (c) del Teorema 5.4). Dunque $v_2 \neq v_0$. Se $v_2 = r$, allora v_2, v_1, v_0 è il cammino cercato. Altrimenti iteriamo: fintantoché $v_i \neq r$, definiamo v_{i+1} come l'unico nodo tale che $(v_{i+1}, v_i) \in A$. La sequenza di nodi v_i così costruita è formata da vertici distinti, perché se si avesse $v_i = v_j$ per qualche $i > j$, allora $v_i, v_{i-1}, \ldots, v_j$ sarebbe un ciclo in D, contro l'ipotesi che G sia un albero. Inoltre, tale sequenza è necessariamente finita, perché V è un insieme finito. Quindi prima o poi avremo $v_i = r$ e dunque $v_i, v_{i-1}, \ldots, v_0$ sarà il cammino orientato cercato.

Per mostrare l'implicazione inversa, supponiamo che D sia un'arborescenza con radice r. Dato un qualunque nodo $v \in V$, per ipotesi esiste un cammino orientato da r a v in D e quindi un cammino (non orientato) tra r e v in G. Allora G è un grafo connesso. Inoltre, essendo G il grafo sottostante a D, si ha $|E| \leq |A| = |V| - 1$. Dalla condizione (d′) del Teorema 5.4 deduciamo che G è un albero. Rimane da dimostrare che $|\delta^-(r)| = 0$ e $|\delta^-(v)| = 1$ per $v \neq r$. Si noti che $|\delta^-(v)| \geq 1$ per ogni $v \neq r$, altrimenti non ci sarebbe nessun cammino orientato da r a v in D. Allora, poiché

$$|V| - 1 = |A| = \sum_{v \in V} |\delta^-(v)| = |\delta^-(r)| + \sum_{v \in V \setminus \{r\}} |\delta^-(v)| \geq |V| - 1,$$

deve valere $|\delta^-(r)| = 0$ e $|\delta^-(v)| = 1$ per ogni $v \neq r$. □

5.2 Complessità degli algoritmi

Un elemento centrale di questo corso è lo sviluppo di algoritmi per risolvere certi problemi di ottimizzazione. Sebbene le definizioni di "problema" e di "algoritmo" possano essere formalizzate attraverso il concetto di *macchina di Turing*, lo sviluppo rigoroso della teoria della complessità computazionale è al di fuori degli obiettivi di queste note; la trattazione che qui proponiamo sarà quindi meno approfondita.

In quanto segue assumeremo sempre di avere a che fare con problemi i cui dati sono numeri razionali. Questa è un'ipotesi standard in teoria della complessità computazionale ed è giustificata dal fatto che in un calcolatore non possiamo codificare numeri irrazionali, in quanto questi non ammettono una rappresentazione finita. Ricordiamo, inoltre, che nei calcolatori tutte le informazioni sono espresse in codice binario.

Introduciamo la distinzione tra problemi e istanze. Un *problema* viene generalmente espresso come un quesito in cui è possibile (e comune) che appaiano dei parametri. Ad esempio, il problema della programmazione lineare in forma standard può essere così formulato: dati due interi positivi m e n, una matrice $A \in \mathbb{Q}^{m\times n}$ e due vettori $b \in \mathbb{Q}^m$ e $c \in \mathbb{Q}^n$, trovare (se esiste) un vettore $x \in \mathbb{Q}^n$ che risolva il problema

$$\max \quad c^{\mathsf{T}}x \tag{5.3}$$

$$\text{s.a} \quad Ax = b \tag{5.4}$$

$$x \geq 0. \tag{5.5}$$

I parametri (o dati) del problema sono m, n, A, b e c. Quando ai parametri vengono assegnati valori specifici, si ottiene un'*istanza* del problema. Un problema è dunque un insieme di istanze, una per ogni possibile scelta dei dati.

Un *algoritmo* per un problema è una sequenza di operazioni che, eseguite su una qualunque istanza del problema, portano alla soluzione dell'istanza. Nella valutazione dell'efficienza degli algoritmi riveste un ruolo cruciale la misura del loro tempo di esecuzione: un algoritmo più veloce sarà in genere preferito ad uno più lento. Spesso ci si riferisce alla velocità di un algoritmo come alla sua *complessità*; la branca dell'informatica teorica che si occupa di questi argomenti è detta *teoria della complessità (computazionale)*. Per definire più rigorosamente i concetti di "veloce" e "lento", dobbiamo approfondire il modo in cui un computer riceve in input un'istanza di un problema.

Un calcolatore riceve in input una data istanza di un problema mediante una *stringa binaria* (cioè di 0 e 1), il cui singolo carattere è detto *bit*. Ad esempio, per memorizzare un numero intero a sono necessari $1+\lceil \log_2(|a|+1)\rceil$ bit:[1] un bit per il segno di a, e $\lceil \log_2(|a|+1)\rceil$ bit per rappresentare $|a|$. Per codificare un numero razionale p/q, dove p è un intero e q un intero positivo, sono dunque necessari $1 + \lceil \log_2(|p|+1)\rceil + \lceil \log_2(|q|+1)\rceil$ bit. Definiamo *grandezza* o *taglia* di un'istanza I (e la indichiamo con $\text{size}(I)$) il numero di bit necessari a codificare tutti i dati dell'istanza.

Una volta ricevuta in input un'istanza di un problema, il computer esegue una serie di operazioni per la sua soluzione. L'unità di misura per valutare il tempo impiegato da un algoritmo è data dal numero di *operazioni elementari* che esso esegue. Le operazioni elementari sono l'addizione, la sottrazione, il prodotto, la divisione ed il confronto tra due numeri: con esse è possibile implementare tutte le operazioni più complesse svolte da un calcolatore. Dati un problema P ed un algoritmo A per P, per ogni istanza $I \in P$ indichiamo con $\text{time}(A, I)$ il numero di operazioni elementari necessarie alla completa esecuzione dell'algoritmo A sull'istanza I. (Scriviamo più brevemente $\text{time}(I)$ quando è chiaro a quale algoritmo ci riferiamo.) È naturale aspettarsi che il numero di operazioni sia tanto più elevato quanto più $\text{size}(I)$ è grande. Possiamo ritenere un algoritmo efficiente quando il numero di operazioni che esso richiede cresce in modo "ragionevole" all'aumentare della grandezza dell'istanza. Diremo allora che A è un *algoritmo polinomiale* per P se esiste un polinomio p (in una variabile) tale

[1] Dato un numero x, indichiamo con $\lceil x\rceil$ l'arrotondamento per eccesso di x, cioè il minimo numero intero maggiore o uguale di x.

n	1	10	100	1000	10000	100000
$10^6 n^2$	10^6	10^8	10^{10}	10^{12}	10^{14}	10^{16}
n^{100}	1	10^{100}	10^{200}	10^{300}	10^{400}	10^{500}
1.1^n	1.1	2.6	10^4	10^{41}	10^{414}	10^{4139}

Tabella 5.1: Crescita delle funzioni $10^6 n^2$, n^{100} e 1.1^n all'aumentare di n (i valori di 1.1^n sono approssimativi). Si noti che nel confronto tra le prime due funzioni, che sono polinomiali, è il grado del polinomio a stabilire quale funzione cresce più rapidamente per $n \to +\infty$, nonostante la prima funzione abbia un coefficiente molto elevato. Inoltre, la terza funzione, che è esponenziale e non polinomiale, asintoticamente cresce più rapidamente di qualunque polinomio, nonostante la sua base sia molto vicina a 1.

che, per ogni istanza $I \in P$, si abbia $\text{time}(A, I) \leq p(\text{size}(I))$. Si verifica che la definizione di algoritmo polinomiale è equivalente alla seguente, di fatto più pratica: A è un algoritmo polinomiale se esistono due costanti c e d tali che per ogni istanza I del problema si abbia $\text{time}(A, I) \leq c \cdot \text{size}(I)^d$. In realtà, saremo interessati unicamente al comportamento asintotico dell'algoritmo all'aumentare di $\text{size}(I)$, ovvero al grado d, mentre la costante c sarà ignorata. Pertanto, se un algoritmo soddisfa la proprietà precedente, diremo che ha *tempo di esecuzione* (o *complessità*) $O\left(\text{size}(I)^d\right)$ (in altre parole, il numero di operazioni richieste all'aumentare della grandezza di I cresce asintoticamente come il polinomio x^d). L'interesse per gli algoritmi polinomiali deriva dal fatto che i polinomi crescono molto più lentamente delle funzioni esponenziali (si veda la Tabella 5.1), che corrispondono generalmente al tempo di esecuzione di algoritmi banali consistenti nell'enumerazione di tutte le possibili soluzioni ammissibili di un problema (ad esempio, l'enumerazione di tutte le basi ammissibili per determinare la soluzione ottima di un programma lineare).

Come si evince dalla definizione, per decidere se un algoritmo è polinomiale è necessario conoscere la grandezza delle istanze $I \in P$. Tuttavia, è in realtà sufficiente saper stimare $\text{size}(I)$ in modo molto approssimativo. Vediamo alcuni esempi.

Esempio 5.7 Consideriamo il seguente problema:

> *Dati due interi positivi m e n, una matrice $A \in \mathbb{Q}^{m \times n}$ e due vettori $b \in \mathbb{Q}^m$ e $c \in \mathbb{Q}^n$, risolvere il programma lineare* (5.3)–(5.5).

Si ottiene un'istanza I di questo problema quando vengono specificati m, n, A, b e c; devono dunque essere codificati due numeri interi (m e n) e $mn + m + n$ numeri razionali. Sia T la grandezza della rappresentazione binaria della componente di A, b e c che richiede il maggior numero di bit. Allora

$$\text{size}(I) \leq \text{size}(m) + \text{size}(n) + (mn + m + n)T. \tag{5.6}$$

Poiché $\text{size}(m) = 1 + \lceil \log_2(m+1) \rceil \leq m + 1$ e $\text{size}(n) = 1 + \lceil \log_2(n+1) \rceil \leq n + 1$, la disuguaglianza (5.6) implica che ogni polinomio in $\text{size}(I)$ è maggiorato da un polinomio in m, n, T. D'altro canto, poiché ogni componente di A, b e c richiede almeno un bit per essere codificata e poiché almeno una delle componenti richiede T bit, si ha

$$\text{size}(I) \geq mn + m + n \quad \text{e} \quad \text{size}(I) \geq T.$$

Questo implica che ogni polinomio in m, n, T è maggiorato da un polinomio in size(I). Possiamo allora affermare che time(I) è limitato da un polinomio in size(I) se e solo se è limitato da un polinomio in m, n, T. Dunque è possibile controllare la polinomialità di un algoritmo per questo problema senza conoscere con precisione la grandezza dell'istanza, ma basandosi unicamente sui valori di m, n e T.

Esempio 5.8 Consideriamo il seguente problema:

> *Dato un grafo G, decidere se esiste un ciclo in G che attraversi tutti i nodi di G (un tale ciclo è detto* ciclo hamiltoniano*).*

In questo caso, ogni istanza I del problema è un grafo G. Chiamiamo n il numero di nodi e m il numero di spigoli di G. Assumiamo che G non abbia nodi isolati (cioè nodi di grado zero), ipotesi che può essere fatta nella grande maggioranza dei problemi su grafi (i nodi isolati possono in genere essere rimossi senza che la soluzione del problema cambi). Si noti che allora il Lemma 5.1 implica $m \geq n/2$. Il modo più ovvio per codificare un grafo in un calcolatore consiste nel registrare il numero n di nodi e l'elenco delle m coppie di nodi che sono unite da uno spigolo. Senza perdita di generalità, i nodi possono essere chiamati $1, \ldots, n$ (senza la necessità di codificare questa informazione); allora ogni spigolo potrà essere rappresentato tramite una coppia di numeri, ciascuno dei quali richiederà un numero di bit compreso tra 1 e size(n). Dunque avremo

$$\text{size}(n) + 2m \leq \text{size}(I) \leq \text{size}(n) + 2m\,\text{size}(n),$$

dove la prima disuguaglianza vale perché n viene codificato con size(n) bit e le m coppie di nodi richiedono almeno due bit ciascuna (uno per ogni elemento della coppia). Ne segue che time(I) è limitato da un polinomio in size(I) se e solo se è limitato da un polinomio in m e n; ma siccome $n/2 \leq m \leq n(n-1)/2$, questo equivale a richiedere che time(I) sia limitato da un polinomio in n. Dunque è possibile controllare la polinomialità di un algoritmo per questo problema senza conoscere con precisione la grandezza dell'istanza, ma basandosi unicamente sul numero di nodi (ed eventualmente di spigoli) del grafo. (Esistono metodi alternativi per codificare un grafo, ma la conclusione a cui si giunge è la stessa.)

Discuteremo più sotto la polinomialità del problema dell'Esempio 5.7. Per quanto riguarda il problema dell'Esempio 5.8 (decidere se un grafo contiene un ciclo hamiltoniano), osserviamo che un algoritmo per risolvere il problema potrebbe essere quello banale di enumerare tutte le $n!$ possibili sequenze di nodi del grafo, verificando di volta in volta se la sequenza in esame formi un ciclo. Tale algoritmo è completamente insoddisfacente, in quanto, all'aumentare del numero di nodi del grafo, il numero di possibili sequenze aumenta più che esponenzialmente. Anche per poche decine di nodi un algoritmo di questo tipo sarebbe inapplicabile: ad esempio, per un grafo di soli 30 nodi, anche se avessimo un computer in grado di generare una possibile sequenza ogni picosecondo (10^{-12} secondi), enumerare tutte le possibili 30! sequenze richiederebbe oltre 8400 miliardi di anni. Non è noto alcun algoritmo polinomiale per risolvere il problema del ciclo hamiltoniano. È opinione quasi universalmente condivisa che un algoritmo polinomiale per tale problema non esista affatto. Tale convinzione si basa su concetti di teoria della complessità computazionale che esulano dagli obiettivi di queste note.

5.2.1 Complessità della Programmazione Lineare

Come visto nel Capitolo 3, il metodo del simplesso (con l'applicazione delle sue due fasi e l'utilizzo di una regola di pivot anti-ciclaggio, come ad esempio quella di Bland) è un algoritmo per il problema della programmazione lineare. È naturale chiedersi se tale algoritmo sia polinomiale.

Restringendoci alla sola Fase II, possiamo dividere lo studio della complessità del metodo del simplesso in due parti: l'esecuzione di una singola iterazione, che prende in input una base ammissibile ed il tableau relativo, sceglie le variabili da far entrare in base ed uscire dalla base, effettua il cambio di base e costruisce il nuovo tableau; e il numero di iterazioni effettuate dall'algoritmo, pari al numero di cambi di base da effettuare per raggiungere l'ottimo (o concludere che l'ottimo non esiste) partendo da una prima base ammissibile.

Osservazione 5.9 *Una singola iterazione del metodo del simplesso può essere eseguita in tempo polinomiale nella grandezza dell'istanza.*

Dimostrazione. Per scegliere l'indice k che entra in base, basta scorrere in ordine le componenti del vettore $\bar{c}_N$ fino ad individuare la prima componente $\bar{c}_k > 0$ (se tale componente non esiste, abbiamo trovato la soluzione ottima): questo richiede meno di n confronti tra numeri. Per scegliere la variabile che esce di base (o concludere che il problema è illimitato), è sufficiente calcolare e confrontare tra loro al massimo m rapporti. L'operazione di pivot consiste nel calcolare le nuove componenti del tableau; le componenti da calcolare sono al massimo $(m+1)(n+1)$ e per ciascuna di esse è sufficiente un numero costante di operazioni elementari (si vedano le formule della Sezione 3.4). Nel complesso, il numero di operazioni elementari eseguite durante una singola iterazione è polinomiale nella grandezza dell'istanza. □

Per l'osservazione precedente, se l'algoritmo del simplesso eseguisse per ogni istanza I un numero di iterazioni polinomiale in size(I), allora la sua complessità sarebbe polinomiale. Poiché il numero di iterazioni dipende dalla scelta delle variabili da far entrare in base ed uscire dalla base, la polinomialità potrebbe dipendere dalla regola di pivot utilizzata. Oltre alla regola di Bland, esistono altre regole che garantiscono che l'algoritmo termini in un numero finito di iterazioni. Tuttavia, per tutte le regole di pivot finora proposte sono stati trovati esempi che mostrano che il metodo del simplesso con tale regola può richiedere un numero di iterazioni dell'ordine di 2^n o $2^{\sqrt{n}}$ (dove n è il numero di variabili del problema), dunque un numero di iterazioni che non può essere limitato da alcun polinomio in size(I). Il primo esempio in questo senso venne proposto da Klee e Minty nel 1972 per mostrare che la regola di Bland non garantisce la polinomialità dell'algoritmo. Un recente esempio di questo tipo, valido per un'altra particolare regola di pivot, è datato 2011. A tutt'oggi non è noto se esista una regola di pivot che garantisca la polinomialità del metodo del simplesso; ciononostante, il comportamento dell'algoritmo nella pratica è piuttosto soddisfacente.

Mentre la questione della polinomialità del metodo del simplesso è ancora irrisolta e rappresenta una dei problemi aperti più importanti della teoria dell'ottimizzazione, esistono invece altri algoritmi polinomiali per la risoluzione di problemi di programmazione lineare. In effetti, l'esistenza di un tale algoritmo è rimasta una questione aperta per decenni, fino a quando nel 1979 Khachiyan trovò il primo algoritmo polinomiale per la programmazione lineare, noto come *metodo dell'ellissoide*. A dispetto del suo interesse teorico, il metodo di Khachiyan non è tuttavia efficiente in pratica. Si dovette attendere l'avvento dei cosiddetti *metodi di punto interno* (primi anni '90) per avere algoritmi per la programmazione lineare

che fossero efficienti sia in teoria (cioè che fossero polinomiali) che nella pratica. I solutori commerciali di problemi di programmazione lineare fanno in genere uso del metodo del simplesso (primale o duale) e dei metodi di punto interno, a seconda del particolare problema in esame (o di un'eventuale richiesta specifica da parte dell'utente).