Formulario di Modelli di Calcolo e Algoritmi Avanzati

Paolo Speziali

Giugno 2022

1 Grammatiche

Tipi di grammatiche

Tipo 0
Detta anche **non limitata**, ammette **qualunque tipo** di produzione, anche quelle che accorciano le forme di frase, come ad esempio quelle che si ottengono da ε-produzioni (es. $aA \to b \mid \varepsilon$).

Tipo 1
Detta anche **context-sensitive**, ammette **qualunque tipo** di produzione che **non accorci** le forme di frase (es. $aA \to Bb$).

Tipo 2
Detta anche **context-free**, ammette produzioni in cui il termine sinistro è formato da un solo non terminale e la forma di frase non si accorcia mai (es. $A \to aBb$).

Tipo 3
Detta anche **regolare**, ammette produzioni in cui il termine sinistro è formato da un solo non terminale e il termine destro o da un solo terminale o da un terminale seguito da un non terminale (es. $A \to b \mid aB$).

2 Macchine a Registri

Operandi ed etichette

Ogni operando `<op>` può avere una delle forme seguenti:

- **n**: `<op>` indica l'intero memorizzato nel registro n;
- **(n)**: `<op>` indica l'intero memorizzato nel registro indirizzato dal registro n;
- **#n**: `<op>` indica il numero n.

Ogni etichetta `<et>` ha la forma **n**, ed indica il numero intero n.

Istruzioni di trasferimento

- **LOAD** <op> (`R[0] := <op>`);
- **STORE** <op> (`R[<op>] := R[0]`).

Istruzioni aritmetiche

- **ADD** <op> (`R[0] := R[0] + <op>`);
- **SUB** <op> (`R[0] := R[0] - <op>`);
- **MULT** <op> (`R[0] := R[0] * <op>`);
- **DIV** <op> (`R[0] := R[0] / <op>`).

Istruzioni di I/O

- **READ** <op> (`R[<op>] := IN`);
- **WRITE** <op> (`OUT := <op>`).

Istruzioni di salto e di controllo

- **JUMP** <et> (`CI := <et>`);
- **JGTZ** <et> (`if (R[0] > 0) then CI := <et> else CI := CI + 1`);
- **JZERO** <et> (`if (R[0] = 0) then CI := <et> else CI := CI + 1`);
- **HALT** (`fine, il calcolo si arresta`).

3 Linguaggi

Pumping Lemma per L. Regolari

Se L è un linguaggio regolare allora
$\exists n > 0$ tale che $\forall z \in L$ e $|z| \geq n$,
si ha che $\exists u, v, w$ tali che:

1. $z = uvw$
2. $|uv| \leq n$
3. $|v| \geq 1$
4. $uv^i w \in L, \forall i \geq 0$

Da Gram. Reg. a Esp. Reg.

Dal sistema di equazioni lineari si ricava una espressione regolare applicando le due tecniche seguenti ripetutamente:

- **Sostituzione**: si può sostituire un simbolo non terminale con una espressione equivalente (es. $A\,aB + b,\ B = cA \Rightarrow = acA + b$)
- **Eliminazione della ricorsione**: si può sostituire la prima equazione con la seconda
  $A = \alpha_1 A + \alpha_2 A + ... + \alpha_n A + \beta_1 + \beta_2 + ... + \beta_n$
  $A = (\alpha_1 + \alpha_2 + ... + \alpha_n)^*(\beta_1 + \beta_2 + ... + \beta_n)$

Forma ridotta

Una grammatica G CF è in forma ridotta se:

- G non contiene ε-produzioni se non sull'assioma, ed in tal caso l'assioma non compare a destra di nessuna produzione;
- G non contiene simboli inutili, cioè non fecondi o non generabili;
- G non contiene produzioni unitarie (cioè del tipo $A \to B$).

Forma normale di Greibach (GNF)

Una grammatica G CF è in GNF se tutte le sue produzioni sono del tipo $A \to a\beta$, dove β è una sequenza (eventualmente vuota) di non terminali e a è un simbolo terminale.

Forma normale di Chomsky (CNF)

Una grammatica G CF è in CNF se tutte le sue produzioni sono del tipo $A \to BC$ o $A \to a$.
Per arrivarci si porta G in forma ridotta e si sostituisce ogni terminale a con un non terminale X_a in tutte le produzioni in cui compare a ed si introduce la produzione $X_a \to a$ (la forma ottenuta a questo punto si chiama **quasi CFN**).
Infine si sotituisce ricorsivamente ogni produzione del tipo:
$A \to BC\alpha$ con le seguenti: $A \to BD$, $D \to C\alpha$ dove D è un nuovo non terminale.

Pumping Lemma per L CF

Se L è un linguaggio CF allora
$\exists\, n > 0$ tale che $\forall\, z \in L$ e $|z| \geq n$,
si ha che $\exists\, u, v, w, x, y$ tali che:

1. $z = uvwxy$
2. $|vwx| \leq n$
3. $|vx| \geq 1$
4. $uv^i wx^i y \in L, \forall\, i \geq 0$

4 Trattabilità

Problemi in NP

Dimostrare che un problema B appartiene ad NP:

1. Definire in cosa consiste l'insieme S delle **soluzioni candidate** di B e quale è la proprietà che ogni soluzione candidata deve possedere per essere una soluzione effettiva;
2. Definire uno **schema di codifica** delle soluzioni candidate di B;
3. Mostrare che le codifiche delle soluzioni candidate hanno **taglia polinomiale** nell'input;
4. Definire un **algoritmo deterministico** che verifica in tempo polinomiale se una data soluzione candidata rispetta la proprietà richiesta.

5 Complessità

Tipologie di problemi

Per definire le varie tipologie di problemi useremo degli esempi che partono dal concetto di **Clique**.

Problemi di decisione

Un prob. di decisione P_D è definito da:

- Un insieme di istanze I_{P_D};
- Un predicato π: $I_{P_D} \to$ {vero, falso}.

`CLIQUE`
`Istanza`: Grafo G, intero $K > 0$
`Predicato`: Esiste una clique di G di dimensione $\geq$ K ?

Problemi di ricerca

Un prob. di decisione P_R è definito da:

- Un insieme di istanze I_{P_R};
- Un insieme di soluzioni candidate S_{P_R};
- Una proprietà di ammissibilità che deve valere per ogni soluzione di una data istanza.

`RICERCA CLIQUE`
`Istanza`: Grafo G, intero $K > 0$
`Soluzione candidata`: Sottoinsieme V' di vertici
`Ammissibilità`: V' è una clique di G di dimensione $\geq$ K
Un algoritmo che lo risolve prende in input una istanza x e restituisce (se esiste) un elemento di $Sol(x) = \{\, y \in S_{P_R} : y$ rispetta la proprietà di ammissibilità $\}$.

Tipologie di problemi

Ogni problema di ricerca P_R ha un problema di decisione associato P_D.

Problemi di enumerazione

Un prob. di decisione P_E è definito da:

- Un insieme di istanze I_{P_E};
- Un insieme di soluzioni candidate S_{P_E};
- Una proprietà di ammissibilità.

Un algoritmo che lo risolve prende in input una istanza x e restituisce la cardinalità dell'insieme $Sol(x)$.

Problemi di ottimizzazione

Un prob. di decisione P_O è definito da:

- Un insieme di istanze I_{P_O}, un insieme di soluzioni candidate S_{P_O}, ed una proprietà di ammissibilità;
- Una funzione di misura $\mu : I_{P_O} \times S_{P_O} \to \mathbb{N}$ definita solo sulle coppie (x, y) tali che $y \in Sol(x)$;
- Un criterio di scelta $c \in$ {MIN, MAX};

Un algoritmo che lo risolve prende una istanza x e restituisce y tale che,
$\forall\, z \in Sol(x)$,
$m(x, y) \leq m(x, z)$ se $c =$ MIN e
$m(x, y) \geq m(x, z)$ se $c =$ MAX.
`MASSIMA CLIQUE`
`Istanza`: Grafo G
`Soluzione candidata`: Insieme V' di vertici
`Ammissibilità`: V' è una clique di G
`Misura`: $|V'|$
`Criterio`: MAX

6 Complessità parametrica

Problema parametrizzato

Si tratta di un linguaggio $L \subseteq \Sigma^* \times \mathbb{N}$. Data un'istanza $(x, k) \in \Sigma^* \times \mathbb{N}$, k è il **parametro** del problema.
L è detto Fixed-Parameter Tractable (**FPT**) se esiste una terna (A, f, c) dove:

- $f : \mathbb{N} \to \mathbb{N}$ è una funzione calcolabile;
- c è una costante;
- A è un algoritmo che decide se una data istanza (x, k) appartiene ad L in un tempo limitato superiormente da $f(k) \cdot |x|^c$.

L è detto Slice-Wise Polynomial (**XP**) se esiste una terna (A, f, g) dove:

- $f, g : \mathbb{N} \to \mathbb{N}$ sono due funzioni calcolabili;
- A è un algoritmo che decide se una data istanza (x, k) appartiene ad L in un tempo limitato superiormente da $f(k) \cdot |x|^{g(k)}$.

Kernelizzazione

Approccio sistematico allo studio di algoritmi di preprocessing che lavorano in tempo polinomiale.
Sia $Q \subseteq \Sigma^* \times \mathbb{N}$ un problema parametrizzato, diciamo che due istanze (x, k) e (x', k') sono equivalenti se $(x, k) \in Q \Leftrightarrow (x', k') \in Q$.
Una funzione $\Phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ è una **regola di riduzione** per Q se:

- Φ mappa ciascuna istanza (x, k) in una equivalente (x', k'), ovvero è **safe**;
- Φ è calcolabile in tempo polinomiale nella dimensione di (x, k).

La **dimensione dell'output** di A è una funzione $size_A(k) : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ definita come segue:
$size_A(k) = sup\{|x'| + k' : (x', k') = A(x, k), x \in \Sigma^*\}$.
Un **algoritmo di kernelizzazione** A per il problema Q è un algoritmo di preelaborazione tale che $size_A(k) \leq g(k)$, per una qualche funzione calcolabile $g : \mathbb{N} \to \mathbb{N}$. Se la funzione g è **polinomiale** (**lineare**) in k, allora diciamo che Q ammette un **kernel polinomiale** (**lineare**).
Un problema parametrizzato Q **è FPT** se e solo se **ammette un algoritmo di kernelizzazione**.

Matching

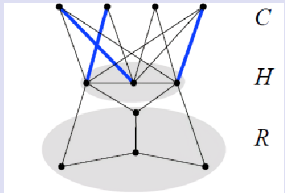
Dato un grafo bipartito $G = (X \cup Y, E)$, un **matching** di X in Y è un insieme $M \subseteq E$ di archi indipendenti tale che ciascun vertice di X è adiacente ad un arco di M.
Per un qualsiasi sottoinsieme $C \subseteq X$, sia $N(C)$ l'insieme di vertici in Y adiacenti ai vertici in C.
I seguenti teoremi rappresentano strumenti fondamentali nell'ambito della kernelizzazione:

- La **dimensione minima di un vertex cover** in un grafo bipartito è pari alla **dimensione del massimo matching**.
- Sia G un grafo bipartito. Allora G **contiene un matching** di X in Y se e solo se $|N(C)| \geq |C|$ per ogni sottoinsieme $C \subseteq X$.
- Sia G un grafo bipartito. Possiamo trovare un matching di dimensione massima (e un vertex cover di dimensione minima) in PTIME.
  Inoltre, in tempo polinomiale, possiamo trovare o un matching di X in Y, oppure un insieme minimale non vuoto $C \subseteq X$ tale che $|N(C)| < |C|$.

Crown Decomposition

Una crown decomposition di un grafo $G = (V, E)$ è una partizione di V in tre insiemi (C, H, R) tale che:

- C è un **indipendent set non vuoto**;
- Nessun vertice di C è connesso a un vertice di R, ovvero **H separa C e R**;
- E **contiene un matching** di H in C.



C = Crown H = Head R = Rest

Sia G un grafo senza vertici isolati e con almeno $3k + 1$ vertici. Allora G contiene un **matching** di dimensione $k + 1$ oppure una **crown decomposition**. Inoltre, una delle due strutture può essere calcolata in tempo polinomiale.

Teoremi di Robertson-Seymour

Un grafo H è un **minor** di G, denotato con $H \leq_M G$, se H può essere ottenuto da un sottografo di G attraverso una serie di contrazioni di un arco (o con rimozione di un arco o un vertice).
Una famiglia di grafi F è **minor-closed**, se per ogni grafo $G \in F$ e per ogni minore H di G, abbiamo che $H \in F$.
$\forall k$ fissato, la famiglia di grafi con vertex cover number al più k è minor-closed.

1. Qualsiasi famiglia infinita di grafi contiene due elementi tali che uno è un minore dell'altro.
   Per ogni fissata famiglia minor-closed F, esiste un insieme finito di grafi $Forb(F)$, detta famiglia dei **minimal forbidden minors**, tale che qualsiasi grafo G appartiene a F se e solo se non esiste un minor di G che corrisponde (a meno di un isomorfismo) a un membro di $Forb(F)$.
2. Esiste una funzione calcolabile f e un algoritmo A, tali che per una data coppia di grafi $H = (V', E')$ e $G = (V, E)$, A decide se $H \leq_M G$ in tempo $f(|H|) \cdot |V|^3$.
3. Sia F una famiglia di grafi minor-closed, esiste una costante c_F che dipende soltanto da F, tale che per ogni grafo G con n vertici, possiamo decidere se $G \in F$ in tempo al più $c_F \cdot n^3$.
4. Per ogni fissato k, esiste un algoritmo che risolve `VERTEX-COVER` in tempo $c_k \cdot n^3$ dove $c_k = f(k)$ per una qualche funzione calcolabile f.

L'ultimo teorema non ci dimostra che `VERTEX-COVER` è FPT, ci dice però che esiste una collezione di algoritmi FPT per il nostro problema, uno per ogni valore di k.
Un problema parametrizzato L è detto **nonuniformly FPT** se esiste una terna $((A_K)_{K \in \mathbb{N}}, f, c)$ dove:

- $f : \mathbb{N} \to \mathbb{N}$ è una funzione calcolabile;
- c è una costante;
- $(A_K)_{K \in \mathbb{N}}$ è una collezione di algoritmi, tale che $\forall k \in \mathbb{N}$, A_k decide se una data istanza (x, k) appartiene ad L in un tempo limitato da $f(k) \cdot |x|^c$.

Treewidth

Se un grafo ha un valore di treewidth piccolo, allora possiamo applicare un approccio di programmazione dinamica simile a quello usato per gli alberi.
Una **tree decomposition** di $G = (V, E)$ è una coppia $\tau = (T, \{X_t\}_{t \in T})$ tale che:

- T è un albero e ciascun nodo $t \in T$ è associato con un un insieme $X_t \subseteq V$ chiamato **bag**;
- $\bigcup_{t \in T} X_t = V$ (per ciascun vertice di G esiste almeno una bag che lo contiene);
- $\forall (u, v) \in E, \exists$ un nodo $t \in T$ tale che $\{u, v\} \subseteq X_t$ (per ciascun arco di G esiste almeno una bag che contiene entrambi i suoi estremi);
- $\forall u \in V$, $T_u = \{t \in T : u \in X_t\}$ induce un sottoalbero connesso di T.

La **width** di $\tau = (T, \{X_t\}_{t \in T})$ è pari alla dimensione della bag più grande meno uno, ovvero:
$width(\tau) = max_{t \in T} |X_t| - 1$.
La **treewidth** di un grafo G, denotata $tw(G)$, è la più piccola width possibile per una sua tree decomposition.
Una tree decomposition τ di G è **nice** se le seguenti proprietà valgono, assumendo che T è radicato nel nodo r:

- $X_r = \varnothing$ e $X_l = \varnothing$ per ogni foglia l di T.
- Per ogni nodo interno t di T, vale una delle seguenti proprietà:
  1. **Introduce node**: t ha solo un figlio t' in T tale che $X_t = X_{t'} \bigcup \{v\}$ per un qualche vertice $v \notin X_{t'}$. Diciamo che v è introdotto in t.
  2. **Forget node**: t ha solo un figlio t' in T tale che $X_t \bigcup \{w\} = X_{t'}$ per un qualche vertice $w \in X_{t'}$. Diciamo che w è dimenticato in t.
  3. **Join node**: t ha solo due figli t_1 e t_1 in T tale che $X_t = X_{t_1} = X_{t_2}$.

Se un grafo G ammette una tree decomposition τ con width al più k, allora ammette anche una nice tree decomposition τ' con width al più k e tale che $|T'| \in O(k|V|)$.
Inoltre, τ' è calcolabile partendo da τ in tempo $O(k^2 \cdot max(|T|, |V|))$.

Path decomposition

Una tree decomposition $\mathrm{P} = (P, X_{t t \in P})$ in cui P è un cammino, prende il nome di **path decomposition**.
La **width** di $\mathrm{P} = (P, X_{t t \in P})$ è ancora pari alla dimensione della bag più grande meno uno, ovvero: $width(\mathrm{P}) = max_{t \in P} |X_t| - 1$.
La **pathwidth** di un grafo G, denotata $pw(G)$, è la più piccola width possibile per una sua path decomposition.
Per definizione, dato un qualsiasi grafo G, abbiamo che $tw(G) \leq pw(G)$.
Una **nice path decomposition** può essere definita similmente a una nice tree decomposition, ignorando il concetto di join node.

Calcolo della treewidth

Sia G un grafo con n vertici e sia k un intero positivo. Esiste un algoritmo che, in tempo $O(k^{O(k^3)} \cdot n)$, o calcola una tree decomposition di G con width al più k, oppure termina concludendo che $tw(G) > k$.

W-hierarchy

Uno strumento utile al fine di identificare quali sono i problemi intrattabili a parametro fissato (se quindi ammettono algoritmi FPT) è la **W-hierarchy**.
Siano A, B due problemi parametrizzati. Una riduzione parametrizzata da A a B è un algoritmo che, per ogni data istanza (x, k) di A, fornisce in output un'istanza (x', k') di B tale che:

1. $(x, k) \in A$ se e solo se $(x', k') \in B$, ovvero le due istanze sono equivalenti;
2. $k' \leq g(k)$, dove g è una qualche funzione;
3. (x', k') è calcolata in tempo $f(k) \cdot |x|^{O(1)}$, dove f è una qualche funzione calcolabile.

Se esiste una riduzione parametrizzata da A a B, e B è un problema FPT, allora A è un problema FPT. Se ne esiste un'altra B a C, altro problema paramterizzato, allora esiste una riduzione parametrizzata da A a C.

Circuiti Booleani

Un **circuito booleano** è un grafo diretto aciclico dove ogni nodo v è etichettato come segue:

- v è un nodo di **input** se il suo indegree è 0;
- v è un nodo di **negazione** se il suo indegree è 1;
- v è un nodo di **and** o di **or** se il suo indegree è almeno 2

Inoltre esiste esattamente un nodo con outdegree 0 etichettato come nodo di **output**.
Dato un circuito booleano, assegnando un valore in $\{0, 1\}$ a ciascun nodo di input e applicando gli operatori logici corrispondenti alle etichette dei nodi del circuito, otteniamo un valore in $\{0, 1\}$ per il nodo in output. Se tale valore è 1 allora diciamo che l'assegnamento in input soddisfa il circuito.
Verificare se un dato assegnamento soddisfa o meno un circuito è fattibile in tempo polinomiale nella dimensione del circuito.
Sappiamo che decidere se un circuito booleano ammette un assegnamento dell'input che lo soddisfa è NP-completo. Al fine di definire una versione parametrizzata del problema, definiamo il **peso di un assegnamento** come il numero di nodi in input a cui viene assegnato il valore 1.
Dato un circuito booleano C e un intero k, il `WEIGHTED-CIRCUIT-SAT` (**WCS**) è il problema di decidere se C ammette un assegnamento che lo soddisfa con peso al più k. Se n è la dimensione del circuito, esistono $O(n^k)$ possibili assegnamenti, ognuno verificabile in tempo polinomiale, pertanto WCS appartiene banalmente alla classe XP (si ritiene non sia FPT).
La **depth** di un circuito booleano è pari alla lunghezza del più lungo cammino tra un nodo di input e il nodo di output.
La **weft** di un circuito booleano è pari al numero massimo di nodi con indegree > 2 in un qualsiasi percorso da un nodo di input al nodo di output.
Sia $C_{t,d}$ la famiglia dei circuiti booleani con weft al più t e depth al più d, e sia WSC[$C_{t,d}$] la restrizione del problema WSC a questa famiglia di circuiti.
Per un dato $t \geq 1$, la classe W[t] è costituita da tutti i problemi parametrizzati P per cui esiste una riduzione parametrizzata da P a WSC[$C_{t,d}$] per un qualche fissato valore di $d \geq 1$.