Insegnamento di algoritmi avanzati

Paradigma greedy

Roberto Posenato

ver. 0.5, 01/10/2009

Sommario

1. Introduzione
2. Dettagli
3. Esempi di applicazione
4. Fondamenti teorici
5. Altri problemi risolvibili con greedy
6. Esercizi

Introduzione

In questa lezione si richiamano i concetti fondamentali del paradigma *greedy*, già studiato e usato nel corso di Algoritmi e strutture dati.

Introduzione

Idea generale

Il paradigma *greedy* prevede di risolvere un problema di ottimizzazione nel seguente modo:

1. La soluzione è costruita in modo incrementale;
2. Ad ogni iterazione, si aggiunge alla soluzione una componente scegliendo un elemento che risulta essere la scelta ottimale in quel momento.

Aspetto cruciale:

- Risolve il problema in modo top-down: ad ogni ciclo si riapplica lo stesso procedimento all'unica sottoistanza possibile;
- La scelta della componente ottimale della soluzione è di tipo locale;
- Un algoritmo greedy determina soluzione ottima se questa contiene le soluzioni ottime dei sottoproblemi.

Dettagli

Pseudocodifica

Il paradigma *greedy* può essere così schematizzato:

Procedura `greedy(X)`

// $X = \{x_1, x_2, \ldots, x_n\}$ sono elementi ammissibili per la soluzione.

1: $S = \emptyset$;
2: Ordina X in base alla funzione obiettivo del problema;
3: **for** $(i = 1; \quad i \leq n; \quad i{+}{+})$ **do**
4: **if** (x_i *può essere aggiunto*) **then** $S = S \cup \{x_i\}$;
5: **endfor**
6: **return** S

Dettagli

Considerazioni

- La funzione obiettivo del problema deve permettere di ordinare gli elementi in modo descrescente rispetto alla loro “appetibilità”.
- La soluzione quindi è costruita in modo incrementale considerando gli oggetti uno alla volta e aggiungendo, ogni volta, quello più appetibile, se possibile.
- È come se, ad ogni passo, si determinasse qual è l’elemento migliore da aggiungere senza considerare le scelte future possibili.
- Un algoritmo *greedy* fornisce una soluzione ottima se si verifica *che la soluzione ottima del problema contiene al suo interno le soluzioni ottime dei sottoproblemi.*

Esempi di applicazione
ALBERO MINIMO DI COPERTURA

PROBLEMA ALBERO MINIMO DI COPERTURA (MIN SPANNING TREE)

DESCRIZIONE: Grafo $G = (V, E)$ non orientato, connesso di ordine n.
Funzione *peso* $w : E \to \mathbf{N}$.

QUESITO: Determinare l'albero minimo di copertura $T = (V, E_T)$:

- albero di n nodi;
- $E_T \subseteq E$;
- peso $w(T) = \sum_{e \in E_T} w(e)$ è minimo.

L'insieme delle soluzione ammissibili è l'insieme degli alberi di ordine n con archi in E.
Qual è la sua cardinalità?

Esempi di applicazione
ALBERO MINIMO DI COPERTURA

Lemma 1

Un albero di copertura di costo minimo di un grafo G contiene gli alberi di copertura di costo minimo dei sottografi di G.

Un algoritmo greedy proposto da Kruskal (1956):

Procedura `kruskal(`G`)`

1: $T = \emptyset$;
2: Ordina gli $|E|$ archi per peso crescente;
3: **for** $(i = 1; \quad i \leq |E|; \quad i{+}{+})$ **do**
4: **if** $(a_i = \{i, j\}$ *non forma un circuito con gli archi in* $T)$ **then**
$T = T \cup \{a_i\}$;
5: **endfor**
6: **return** T

Esempi di applicazione
ALBERO MINIMO DI COPERTURA

Fase di verifica di presenza circuito:

- Si svolge in modo efficiente se si usa una struttura dati per *insiemi disgiunti*:
  - Si consideri la struttura *foreste di insiemi disgiunti* dove le unioni sono per *rango* e si esegue la *compressione del cammino*.
  - Se ci sono n operazioni di MARK-SET e m operazioni di MARK-SET, FIND-SET e UNION ($|\{UNION\}| \leq n-1$), la complessità nel caso peggiore è $O(m\,\alpha(n))$, dove $\alpha(n)$ è una variante della funzione inversa di Ackermann ed è ≤ 4 di fatto.

Esempi di applicazione
ALBERO MINIMO DI COPERTURA

Quindi:

- Ogni nodo forma un insieme disgiunto.
- Ad ogni ciclo:
  - Si determinano gli insiemi disgiunti degli estremi dell'arco considerato (FIND-SET);
  - Se non sono uguali, allora l'arco connette due componenti disgiunte, quindi si inserisce nell'albero e si uniscono le componenti (UNION)
  - altrimenti si ignora in quanto unirebbe nodi già raggiungibili creando un ciclo.

Esempi di applicazione
ALBERO MINIMO DI COPERTURA

Analisi complessità:

- La fase di ordinamento è semplice da risolvere e ha costo $O(|E| \log |E|)$.
- Ci sono $|V|$ operazioni di MARK-SET.
- Ci sono $O(|E|)$ operazioni di FIND-SET e UNION.
- Costo totale ciclo for (righe 3-5) è $O((|E| + |V|)\alpha(|V|))$.
- La complessità complessiva è

$$O(|E| \log |E| + (|E| + |V|)\alpha(|V|)) = O(n^2 \log n),$$

dove $n = |V|$.

Esempi di applicazione

Cammini minimi singola sorgente

PROBLEMA CAMMINI MINIMI SINGOLA SORGENTE

DESCRIZIONE: Grafo $G = (V, E)$ orientato e connesso di ordine n.
Funzione *peso* $w : E \to \mathbf{N}$.
Un nodo $x \in V$.

QUESITO: Determinare i cammini minimi da x verso tutti gli altri nodi.

Teorema 1 (Sottostruttura ottima)

Dato un cammino minimo $p = [v_1, v_2, \ldots, v_k]$ *dal vertice* v_1 *al vertice* v_k, *per qualsiasi* $i, j \quad 1 \leq i \leq j \leq k$, *il sottocammino* $p_{ij} = [v_i, v_{i+1}, \ldots, v_j]$ *è un cammino minimo da* i *a* j.

Esempi di applicazione

Cammino minimo singola sorgente

Un algoritmo greedy è stato proposto da Dijkstra ['dɛik,stra] (1959):

- Si inizializza una stima del cammino minimo d per ciascun nodo a $+\infty$. $d[x] = 0$ per definizione.
- Si mantiene un insieme S di nodi per i quali si è già ottenuto un percorso minimo.
- Ad ogni passo, si considera un vertice $u \in V \setminus S$ con la stima del cammino minimo *minima*.
- Si aggiunge u a S e si rilassano (aggiornano) le stime del cammino minimo per tutti gli adiacenti di u.

Esempi di applicazione

Cammino minimo singola sorgente

Due sono gli aspetti da approfondire:

1. Stima del cammino minimo:
   - Per ogni nodo si deve mantenere un limite superiore alla lunghezza del cammino minimo dal nodo iniziale: $d[v]$.
   - Inizialmente $d[v] = +\infty \quad \forall v \in V \setminus \{x\}, \quad d[s] = 0$.
2. Rilassamento:
   - Quando si aggiunge un nodo u ad un cammino minimo, si deve verificare se, passando per u, è possibile migliorare le stime di cammino minimo precedentemente determinate per tutti i suoi adiacenti.
   - Se $(u, v) \in E$ e $(d[v] > d[u] + w(u, v))$, allora $d[v] = d[u] + w(u, v)$.

Esempi di applicazione

Cammino minimo singola sorgente: pseudocodifica

Procedura `Dijkstra`(G, s)

1: d.**insert**$(s, 0)$; // d è una coda con priorità!
2: **foreach** $v \in V \setminus \{s\}$ **do** d.**insert**$(v, +\infty)$;
3: $S = \emptyset$;
4: **while** $(S \subset V)$ **do**
5: $u = d$.**removeMin**$()$; $S = S \cup \{u\}$;
6: **foreach** (v *adiacente a* u) **do**
7: **if** $(d$.**valueOf**$(v) > (d$.**valueOf**$(u) + w(u, v))$ **then**
d.**setValue**$(v, d$.**valueOf**$(u) + w(u, v))$
8: **endfch**
9: **endw**
10: **return** d;

Esercizio: completare l'algoritmo in modo che restituisca anche i cammini minimi oltre i valori minimi.

Esempi di applicazione

Cammino minimo singola sorgente: analisi

- Per la dimostrazione di correttezza si rimanda al Cormen.
- Per il costo in tempo:
  - Ci sono $|V|$ cicli while e $|E|$ cicli for *in totale*, in quando ogni nodo viene visitato una volta sola e quindi anche tutti i suoi archi.
  - Ci sono quindi $|V|$ operazioni di **insert**, $|V|$ operazioni di **removeMin** e $|E|$ operazioni di **setValue**.
  - Se la coda è implementata con un array, **setValue** e **insert** costano $O(1)$ ciascuna mentre **extractMin** $O(|V|)$.
  - Costo totale $O(|V| + |V|^2 + |E|) = O(|V|^2)$.
  - Si possono usare altre strutture dati, come min-heap binario o o heap di Fibonacci: cambia la complessità solo per grafi sparsi, ma non nel caso peggiore.

Fondamenti teorici

Matroidi

La teoria dei matroidi permette di stabilire se la struttura di un problema soddisfa le condizioni sufficienti affinché una soluzione greedy sia ottima.

Definizione 1 (Matroide)

Un matroide è una coppia ordinata $M = (S, \mathfrak{I})$ che soddisfa le seguenti condizioni:

1. S è un insieme finito;
2. $\mathfrak{I}$ è un insieme non vuoto di sottoinsiemi di S tali che $B \in \mathfrak{I} \wedge A \subseteq B \Rightarrow A \in \mathfrak{I}$; I sottoinsiemi sono detti indipendenti di S. $\mathfrak{I}$ è detto ereditario. *Simile al concetto di chiusura di una classe di complessità!*
3. Proprietà di scambio: $A \in \mathfrak{I}, B \in \mathfrak{I} \wedge |A| < |B| \Rightarrow \exists x \in B \setminus A$ tale che $A \cup \{x\} \in \mathfrak{I}$.

Fondamenti teorici

Matroidi

Esempio 1 (Matroide grafico)

Dato un grafo non diretto $G = (V, E)$, il matroide grafico $M_G = (S_G, \mathfrak{I}_G)$ è definito se

- $S_G = E$;
- Se $A \subseteq E$, allora $A \in \mathfrak{I}_G$ sse A è aciclico. Ovvero, $\mathfrak{I}_G$ contiene tutti i sottoinsiemi di archi che inducono su G, presi uno alla volta, una foresta.

Dimostrazione.

1. $S_G = E$ è un insieme finito perché consideriamo solo grafi finiti.
2. $\mathfrak{I}_G$ è ereditario in quanto ogni sottoinsieme di una foresta è una foresta, quindi appartiene a $\mathfrak{I}$.

[continua...] □

Fondamenti teorici

Proprietà dei matroidi

Dimostrazione.

3. Proprietà di scambio
   - Si considerino due foreste $G_A = (V, A)$ e $G_B = (V, B)$ con $|A| < |B|$.
   - Si dimostra facilmente che una foresta di $|V|$ nodi e con k archi contiene esattamente $|V| - k$ alberi.
   - G_B ha meno alberi di $G_A \Rightarrow$ esiste qualche T in G_B i cui vertici sono in due differenti alberi di G_A.
   - Si consideri allora un coppia di nodi connessi u, v di T che stanno in due alberi distinti in G_A.
   - Se in G_A si aggiunge l'arco (u, v), i due alberi di G_A contenenti u e v si fondono in un nuovo albero senza formare un ciclo (i nodi sono presenti una ed una sola volta!).
   - Quindi $A \cup \{(u, v)\} \in \mathfrak{I}$.

□

Fondamenti teorici

Matroidi

Ancora poche definizioni...

- Un elemento $x \notin A$ è un'estensione di $A \in \mathfrak{I}$ se può essere aggiunto preservando l'indipendenza di A.
- Un sottoinsieme $A \in \mathfrak{I}$ è massimale se non ammette estensioni.
- Si dimostra che tutti i sottoinsiemi massimali di un matroide hanno la stessa dimensione (si applica banalmente la proprietà di scambio su un insieme massimale per ottenere un assurdo).
- Se a un matroide è associata una funzione peso $w : S \to \mathbf{N}^+$, allora il matroide è pesato. Il peso di un sottoinsieme A è $w(A) = \sum_{x \in A} w(x)$.

Matroide grafico (2)

Un matroide grafico ammette come sottoinsiemi massimali gli alberi di ordine $|V| - 1$, detti alberi di connessione.

Fondamenti teorici

Algoritmi greedy in un matroide pesato

PROBLEMA MASSIMO DI UN MATROIDE PESATO

DESCRIZIONE: Un matroide pesato $M = (S, \mathfrak{I})$.

QUESITO: Determinare un sottoinsieme di peso massimo $A \in \mathfrak{I}$, detto anche sottoinsieme ottimo.

- Dato che $w()$ è solo positiva, il problema chiede di determinare un sottoinsieme massimale di peso massimo.
- Visto che un qualsiasi sottoinsieme di un sottoinsieme massimale appartiene a $\mathfrak{I}$, è sempre possibile iniziare da un insieme vuoto e aggiungere, ad ogni passo, l'elemento più "pesante" in quel istante fino ad ottenere un sottoinsieme massimale (proprietà dello scambio).
- Ma questo. . . è il paradigma greedy!

Fondamenti teorici

Algoritmi greedy in un matroide pesato

Procedura `greedy`(M, w)

1: $A = \emptyset$; // è insieme indipendente
2: Ordina S in ordine decrescente rispetto a w;
3: **for** $(i = 1; \quad i \leq n; \quad i{+}{+})$ **do**
4: **if** $(A \cup \{s_i\} \in \mathfrak{I})$ **then** $A = A \cup \{s_i\}$;
5: **endfor**
6: **return** A

- Invariante di ciclo: A è indipendente. Slide successiva si dimostra che è massimale e ottimo.
- Il costo in tempo è dato dalla somma:
  - costo di ordinamento: $O(n \log n)$ dove $n = |S|$.
  - $n \times$ costo verifica$(A \cup \{s_i\} \in \mathfrak{I})$ (si assume $= f(n)$) $\Rightarrow O(nf(n))$.

  Totale $O(n \log n + nf(n))$.

Fondamenti teorici

Algoritmi greedy in un matroide pesato

Lemma 2 (Unicità dell'estensione)

Se un elemento $x \in S$ non è un'estensione di $\emptyset$, allora x non è un'estensione di qualsiasi sottoinsieme indipendente di A di S.

Dimostrazione.

Si dimostra semplicemente dimostrando che vale la contronominale: *Se x è un'estensione di qualche sottoinsieme indipendente A, allora x è anche estensione di $\emptyset$.*
Lasciata per esercizio. □

Questo implica che se un elemento non può essere aggiunto quando considerato, non potrà mai essere aggiunto (né prima, né dopo).

Fondamenti teorici

Algoritmi greedy in un matroide pesato

Lemma 3 (Elemento massimale è nella soluzione massimale)

Se si ordina in modo descrescente S e x è il primo elemento di tale ordine tale che $\{x\}$ è indipendente (se esiste), allora esiste un sottoinsieme ottimo A che lo contiene.

Dimostrazione.

- Sia B un sottoinsieme ottimo t.c. $x \notin B$.
- Si costruisca A partendo da $A = \{x\}$ e usando la proprietà di scambio con B.
- Quando $|A| = |B|$, allora $A = B \setminus \{y\} \cup \{x\}$ per un $y \in B$. Ma $w(y) \leq w(x)$ per ipotesi.
  Quindi $w(A) \geq w(B)$.
  Quindi anche A è ottimo. □

Fondamenti teorici

Algoritmi greedy in un matroide pesato

Teorema 2 (Proprietà della sottostruttura ottima dei matroidi)

Sia x il primo elemento di S scelto dalla procedura greedy con input $M = (S, \mathfrak{I})$. Il problema di trovare un sottoinsieme di peso massimo che contiene x si riduce a trovare un sottoinsieme di peso massimo per il sottomatroide $M' = (S', \mathfrak{I}')$

$$S' = \{y \in S \mid \{x, y\} \in \mathfrak{I}\}$$

$$\mathfrak{I}' = \{B \subseteq S - \{x\} \mid B \cup \{x\} \in \mathfrak{I}\}$$

e la funzione peso è adeguata di conseguenza.

Dimostrazione.

- Se A è sottoinsieme indipendente per M, allora $A' = A \setminus \{x\}$ è sottoinsieme indipendente per M' e viceversa.
- Vale $w(A) = w(A') + w(x)$. Se A o A' ha peso massimo, anche l'altro insieme deve avere peso massimo nel proprio matroide.

□

Fondamenti teorici

Algoritmi greedy in un matroide pesato

Teorema 3 (Correttezza dell'algoritmo greedy con i matroidi)

Sia $M = (S, \mathfrak{I})$ un matroide pesato con funzione peso w, allora greedy(M, w) determina un sottoinsieme ottimo.

Dimostrazione.

- Per il lemma 2, l'algoritmo non pregiudica la determinazione della soluzione considerando gli elementi una sola volta.
- Per il lemma 3, l'algoritmo non sbaglia aggiungere l'elemento massimo nella soluzione perché esiste una soluzione ottima che lo contiene.
- Il ciclo **for** dell'algoritmo poi non è altro che l'applicazione del Teorema 2: una volta aggiunto l'elemento più grande, al ciclo successivo si considerano gli elementi rimanenti e quindi si cerca di risolvere il problema per il sottomatroide M'. □

Fondamenti teorici

Applicazione della teoria dei matroidi

PROBLEMA PROGRAMMAZIONE LAVORI DI DURATA UNITARIA SU SINGOLO PROCESSORE

DESCRIZIONE: $S = \{a_1, a_2, \ldots, a_n\}$ lavori di durata unitaria.
$D = \{d_1, \ldots, d_n \mid 1 \leq d_i \leq n\}$ scadenze dei lavori.
$W = \{w_1, \ldots, w_n \mid w_i \geq 0\}$ penalità da pagare. Si paga un costo w_i se il task a_i non termina entro il tempo d_i.

QUESITO: Determinare un piano di programmazione per singolo processore che minimizza le penalità totali.

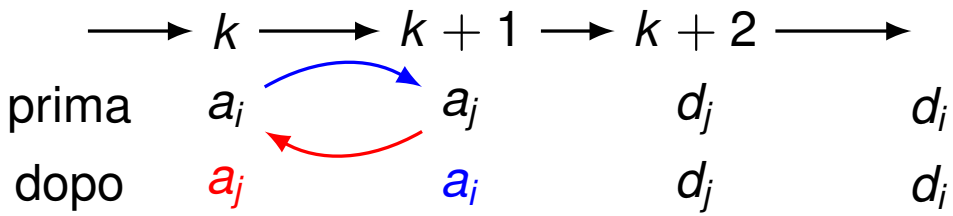
Terminologia e osservazioni:

- Un lavoro è in ritardo se termina dopo la scadenza, in anticipo altrimenti.
- Un piano generico può essere sempre riordinato elencando prima le attività in anticipo e poi quelle in ritardo.

Fondamenti teorici

Applicazione della teoria dei matroidi

- Se le attività in anticipo sono anche ordinate in ordine crescente di scadenza, allora la programmazione è in forma canonica.
  - Si considerino 2 attività in anticipo a_i e a_j che terminano agli istanti k e $k+1$.
  - a_j in anticipo $\Rightarrow k+1 \leq d_j$.
  - Se $d_j < d_i$, allora $k+1 < d_i$ e quindi scambiando a_i con a_j le due attività rimangono ancora in anticipo.



Fondamenti teorici

Applicazione della teoria dei matroidi

- Il problema può essere visto come un problema di determinazione di un opportuno sottoinsieme ottimo delle attività in anticipo!
- Una volta trovato, elencando le attività in anticipo in ordine cronologico e le attività in ritardo in qualsiasi ordine si ottiene la programmazione cercata.
- Un insieme di lavori è indipendente se nessun lavoro è in ritardo;
- Sia $\mathfrak{I}$ l'insieme di tali sottoinsiemi.
- Si indichi con $N_t(A)$, $t = 1, \ldots, n$, il numero di lavori di A la cui scadenza è $\leq t$.
- Si dimostra che A è indipendente sse $N_t(A) \leq t$, $t = 1, \ldots, n$.

Fondamenti teorici

Applicazione della teoria dei matroidi

Il problema di minimizzare la somma delle penalità dei lavori in ritardo è equivalente al problema di massimizzare la somma delle penalità dei lavori in anticipo.

Teorema 4

Sia S l'insieme dei lavori unitari con penalità e $\mathfrak{I}$ l'insieme di tutti gli insiemi indipendenti dei lavori. Allora $M = (S, \mathfrak{I})$ è un matroide.

Fondamenti teorici

Applicazione della teoria dei matroidi

Dimostrazione.

- Dimostriamo solo la proprietà di scambio, lasciando la dimostrazione delle altre due proprietà per esercizio.
- Siano B e A due sottoinsiemi indipendenti con $|B| > |A|$. Sia k il più grande t tale che $N_t(B) \leq N_t(A)$.
- Poiché $N_n(B) = |B|$, $N_n(A) = |A|$ e $|B| > |A|$, deve valere che $k < n$ e $N_j(B) > N_j(A)$ per tutti i $k + 1 \leq j \leq n$. Quindi B contiene uno o più lavori con scadenza $k + 1$ di A.
- Sia a_i un lavoro in $B \setminus A$ con scadenza $k + 1$ e sia $A' = A \cup \{a_i\}$. Per $k < t \leq n$ si ha che $N_t(A') \leq N_t(B) \leq t$ dato che B è indipendente. Dunque A' è indipendente.

Fondamenti teorici

Applicazione della teoria dei matroidi

- Si può quindi applicare il paradigma greedy per risolvere il problema.
- Il costo in tempo è $O(n^2)$ perché ci sono n cicli ciascuno dei quali deve verificare l'indipendenza del sistema, che richiede $O(n)$ passi.

Fondamenti teorici

Limiti dei matroidi

- I problemi sui grafi visti precedentemente sono problemi su matroidi grafici pesati.
- Esistono però anche problemi la cui struttura non è rappresentabile da un matroide ma che ammettono soluzioni greedy.
- L'esempio più interessante di tali problemi è il *problema della compressione di file*, risolvibile in tempo polinomiale con l'algoritmo di Huffman.

Altri problemi risolvibili con greedy

Elenco incompleto di problemi che ammetto un buon algoritmo greedy di risoluzione.

Problema	**Costo naïve**	**Costo greedy**
Albero minimo di ricoprimento		Algoritmo di Prim: $O(n^2)$
Min scheduling di programmi in ritardo	$O(n^2)$	Algoritmo di Moore: $O(n \log n)$
Compressione di file		Codice di Huffman: $O(n \log n)$
Soddisfacibilità delle clausole di Horn (disgiunzioni con al più un letterale positivo)	$O(2^n)$	$O(n)$

Esercizi

Matroidi e minimizzazione

Esercizio 1

Spiegate come dovrebbe essere trasformata la funzione peso di un matroide se si vuole trovare una soluzione di peso minimo per renderlo un problema standard di matroide pesato.

Esercizi

Programmazione lavori di durata unitaria

Esercizio 2

Dimostrare che un insieme A è indipendente se e solo se $N_t(A) \leq t$ e che è possibile verificare l'indipendenza in tempo $O(|A|)$.

Esercizi

Resto in monete

Esercizio 3

Il problema del resto in monete consiste nel formare il resto di n ¢ usando il minor numero possibile di monete. Si assume che il valore delle monete sia un intero.

1. Se le monete sono da 1, 5, 10 e 20 ¢, dimostrare che esiste un algoritmo greedy che determina la soluzione ottima.
2. Lo stesso se le monete sono $c^0, c^1, \ldots, c^k$ ¢ con $c > 1$ e $k \geq 1$.
3. Definite un insieme di monete per il quale l'algoritmo greedy può non determinare la soluzione ottima. L'insieme deve includere la moneta da 1 ¢ per garantire l'esistenza della soluzione.