Insegnamento di algoritmi avanzati

Backtracking

Roberto Posenato

ver. 0.5, 01/10/2009

Sommario

Introduzione

In questa lezione si richiamano i concetti fondamentali della tecnica *backtracking*.

Non si parla di paradigma o di metodo in quanto non vi è uno schema riconoscibile per scrivere un algoritmo di tipo *backtracking* come, invece, si ha per il paradigma *divide et impera* o greedy.

Introduzione

Idea generale

- In taluni casi un approccio *risolvi per tentativi* può risultare conveniente perché è semplice e le alternative, più difficili, non sono più efficienti.
- L'idea del *risolvi per tentativi* è di costruire una soluzione eseguendo un certo numero di azioni e, se non funziona, tornare indietro (backtrack) e eseguire altre azioni.
- Il backtracking è alla base di molti algoritmi di visita di grafi ed è usata per generare sistematicamente tutte le soluzioni ammissibili.

Introduzione

Idea generale

Più in generale:

- Lo spazio delle soluzioni ammissibili può essere rappresentato come un grafo: i nodi sono soluzioni parziali o ammissibili e gli archi rappresentano una qualche dipendenza tra le soluzioni.
- Risolvere il problema equivale ricercare uno specifico nodo/cammino/pattern nel grafo associato.
- Se il grafo ha una dimensione *piccola* rispetto alla dimensione dell'input, tecniche quali la breadth-first o depth-first search sono sufficienti per risolvere il compito.
- Viceversa, se il grafo ha dimensione esponenziale o non limitata, usare tecniche classiche per l'esplorazione è praticamente impossibile.

Introduzione

Idea generale

- In questi casi si preferisce ricorrere a tecniche alternative che usano il cosiddetto grafo implicito: tecniche che esplorano il grafo delle soluzioni ammissibili costruendo, istante per istante, solo la parte di interesse senza mantenere tutto il resto.
- Nella forma più semplice, backtracking può essere descritto come *depth-first search* su un grafo implicito.

Dettagli

Tecnica del backtracking:

- Solitamente il grafo implicito è un albero o, almeno, un DAG non noto a priori.
- Ogni mossa lungo il grafo corrisponde ad aggiungere un nuovo elemento alla soluzione.
- La ricerca ha successo se, procedendo in questo modo, si arriva a determinare una soluzione ammissibile.
- La ricerca si può quindi fermare (se è sufficiente) o continuare (se si vogliono determinare tutte le soluzioni ammissibili o quelle ottimali).
- Se, invece, si arriva a un nodo per cui non è possibile completare la soluzione, si torna indietro (backtrack) fino al primo nodo che ha ancora dei vicini non visitati, verso i quali si riprende la ricerca.

Dettagli

Aspetti cruciali

- Il grafo implicito deve poter essere costruibile in modo incrementale: i vicini di un nodo devono essere definiti in base al nodo e non a condizioni globali;
- Per i problemi di ottimizzazione, la tecnica di backtracking è utile solo per generare in modo ordinato le soluzioni ammissibili (enumerazione). Per cercare quelle ottimali si deve preferire la tecnica *branch & bound* che è un'estensione di questa.

Dettagli

Pseudocodifica

Assunzioni per poter rappresentare un semplice schema di un algoritmo backtracking:

- le soluzioni possono essere rappresentate da un vettore v di dimensione al più n.
- è possibile verificare se una soluzione parziale può essere parte di una soluzione ammissibile: $v[1, \ldots, k], k \leq n$ è detto k-promettente.

Dettagli

Pseudocodifica

Procedura `backtrack(`v`)`

// v è un vettore k-promettente.

1: **if** (`èSoluzione(`v`)`) **then**
2: **return** v;
3: **else**
4: **foreach** $(w[1,\ldots,k+1]\ k+1\text{-}promettente | w[1,\ldots,k]==v[1,\ldots,k])$ **do**
5: `backtrack(`w`);`
6: **endfch**
7: **endif**

- L'algoritmo termina perché non esistono soluzioni $n+1$-promettenti.
- Se ci sono soluzione che sono estensioni di altre, allora si deve rimuovere l'**else**, modificare il **return** in **output** ed eseguire sempre il **for**.

Esempi di applicazione

Problema dello ZAINO

PROBLEMA ZAINO (KNAPSACK)

DESCRIZIONE: Un insieme di n di oggetti caratterizzati ciascuno da un valore v_i e da un peso p_i, entrambi interi positivi; un intero positivo P, portata massima dello zaino.

QUESITO: Determinare un sottoinsieme $X = \{x_1, x_2, \ldots, x_n\}$, con $x_i = 0$ o 1, degli oggetti tale che il valore totale $\sum_{i=1}^{n} x_i v_i$ sia massimo e il peso totale $\sum_{i=1}^{n} x_i p_i \leq P$.

Variante del problema: ZAINO con ripetizione

- Anziché n oggetti si assume di avere n tipi di oggetti;
- Per ciascun *tipo* sono disponibili più oggetti;
- Si richiede quindi di riempire lo zaino anche con più oggetti dello stesso tipo massimizzando il valore totale.

Esempi di applicazione

Problema dello ZAINO con ripetizione

Tecnica del backtracking per lo ZAINO con ripetizione

- Senza perdere in generalità, si considerano i tipi degli oggetti in ordine crescente di peso;
- L'albero implicito è costituito dalla rappresentazione dell'azione di riempimento incrementale dello zaino considerando gli oggetti in ordine crescente di peso.
  - Alla radice lo zaino è vuoto;
  - Ogni nodo figlio rappresenta l'aggiunta di un oggetto.
  - I figli sono considerati in ordine rispetto al peso dell'oggetto che rappresentano.
  - Il primo figlio rappresenta l'aggiunta di un oggetto con tipo uguale all'ultimo aggiunto al padre (le soluzioni sono visitate in modo ordinato).
- Si visita l'albero implicito in modo depth-first.

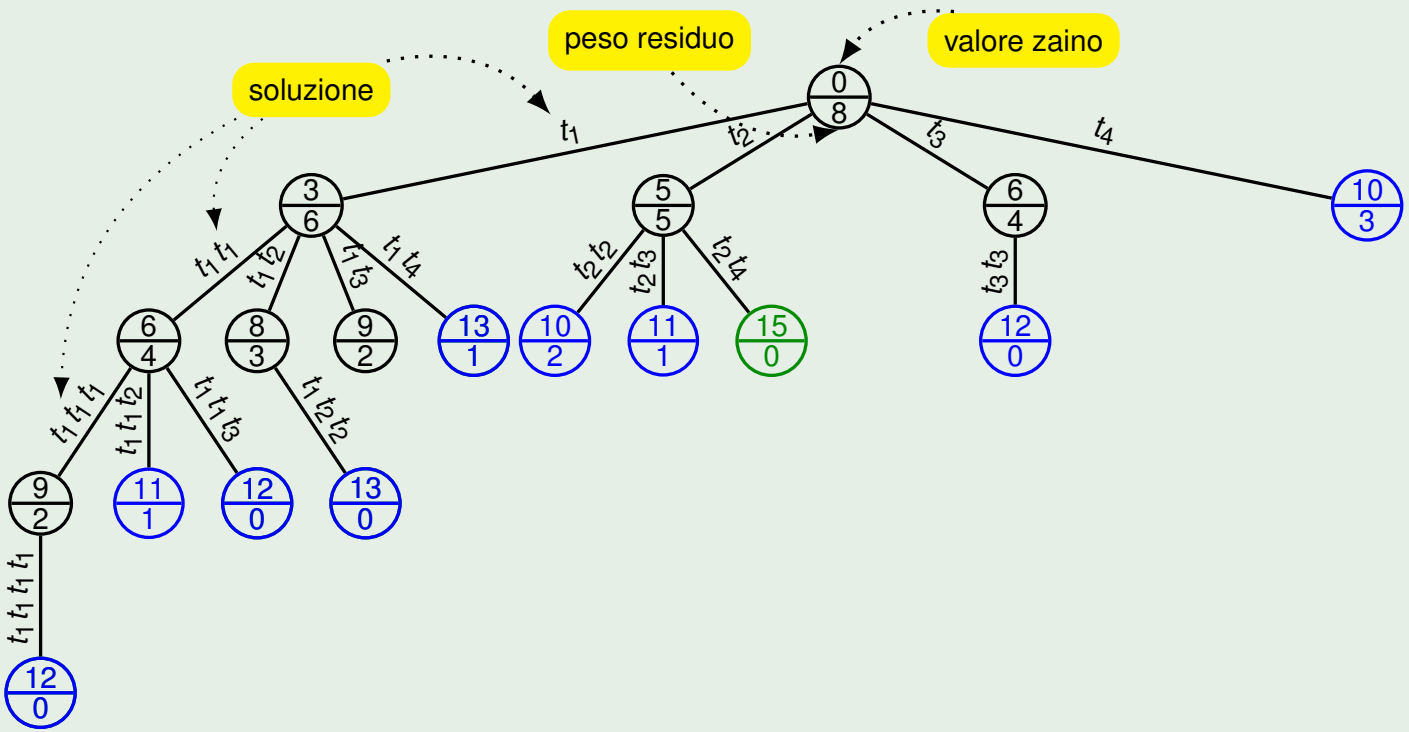
Esempi di applicazione

Problema dello ZAINO con ripetizione

Esempio 1 (Risoluzione di un'istanza)

4 tipi di oggetti, t_1, t_2, t_3, t_4, con pesi $p = [2, 3, 4, 5]$ e valore $v = [3, 5, 6, 10]$; capacità dello zaino $P = 8$.

Albero implicito delle soluzioni:



Esempi di applicazione

Problema dello ZAINO con ripetizione

Altri ordini sono possibili $\Rightarrow$ forma dell'albero diversa.

Funzione `backZaino`(p, v, i, r)

```
// p e v da istanza, i = tipo corrente, r = capacità residua.
1: b = 0;
2: for (k = i;  k < p.length;  k++) do // Prova ogni tipo k residuo
3:     if (p[k] ≤ r) then
       b = max(b, v[k] + backZaino(p, v, k, r − p[k]));
4: endfor
5: return b
```

Esercizio 1

Come si attiva?

Come si deve modificare affinché restituisca anche la soluzione ottima?

Esempi di applicazione

Problema dello ZAINO con ripetizione

Correttezza

- Lo zaino non ammette soluzioni se tutti i pesi degli oggetti sono maggiori della capacità.
  In questo caso l'algoritmo restituisce 0 in quanto il ciclo **for** non aggiunge nessun elemento.
- Per assurdo dimostriamo il caso generale.
- Esiste una soluzione $s = [t_i, t_j, \ldots, t_k]$ ottima che non è stata determinata dall'algoritmo.
- È sempre possibile riodinare gli elementi di s in ordine crescente di peso: $s' = [t_{i_1}, t_{i_2}, \ldots, t_{i_k}]$.

Esempi di applicazione

Problema dello ZAINO con ripetizione

Correttezza

- Si supponga che l'algoritmo abbia aggiunto fino all'elemento $t_{i_j} \quad 1 \leq j < k$ e non abbia aggiunto l'elemento $t_{i_{j+1}}$.
- Questo significa che alla chiamata con $i = i_{j+1}$, nel ciclo **for** l'elemento $t_{i_{j+1}}$ non è stato aggiunto.
- Ciò può accadere se e solo se la soluzione ottimale già determinata ha valore maggiore di quella contenente $t_{i_{j+1}}$. Da cui l'assurdo.

Esempi di applicazione

Problema dello ZAINO con ripetizione

Complessità

- L'algoritmo visita tutte le possibili combinazioni lecite di oggetti nello zaino.
- Il caso peggiore si ha quando tutti i tipi hanno peso unitario.

Esercizio 2

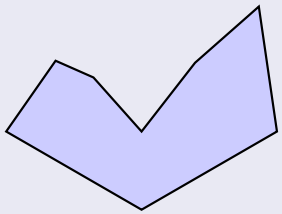
Se n sono i tipi e P è la capacità, quanti nodi ci sono nell'albero?
Suggerimento: si provi a contare il numero di nodi per livello.

Esempi di applicazione

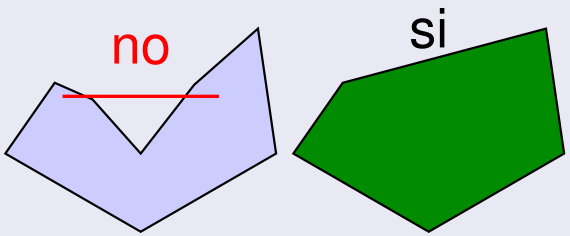
Inviluppo convesso

Prima di presentare il problema, ricordiamo poche definizioni:

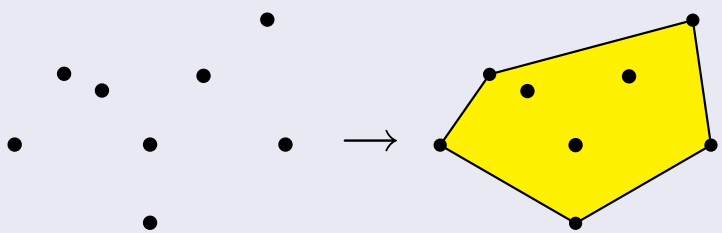
Un poligono è una forma geometrica piana delimitata da una linea spezzata chiusa non intrecciata.



Un poligono è convesso se ogni segmento che congiunge due punti del poligono sta interamente nel poligono stesso.



Dati n punti di un piano, il loro inviluppo convesso è il più piccolo poligono convesso che li contiene.



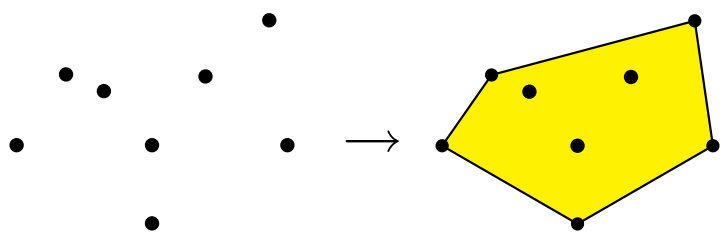
Esempi di applicazione

Inviluppo convesso

PROBLEMA INVILUPPO CONVESSO (CONVEX HULL)

DESCRIZIONE: n punti di un piano.

QUESITO: Determinare il più piccolo poligono convesso che contiene tutti i punti.

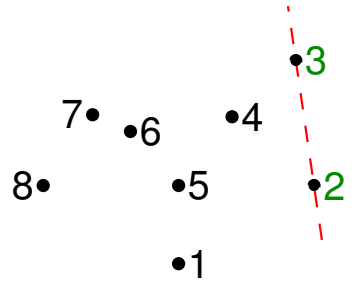


Esempi di applicazione

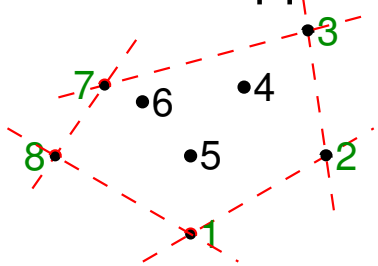
Inviluppo convesso: algoritmo semplice

Un semplice algoritmo determina l'inviluppo convesso sfruttando una proprietà geometrica basilare:

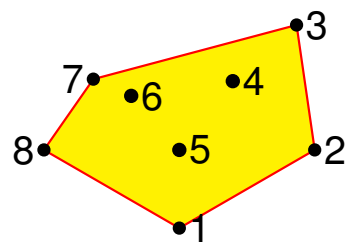
- Dati due punti p_2 e p_3, la retta individuata da questi due punti divide il piano in due semipiani chiusi.



- Se tutti i rimanenti $n-2$ punti giacciono sul medesimo semipiano, allora il segmento $\langle p_i, p_j \rangle$ è uno spigolo dell'inviluppo.
- Si ripete quindi il procedimento per tutte le $\frac{n(n-1)}{2} - 1$ coppie di punti rimanenti.



- Individuati gli spigoli, il poligono è dato elencando i vertici in senso antiorario: $[p_1, p_2, p_3, p_7, p_8]$.

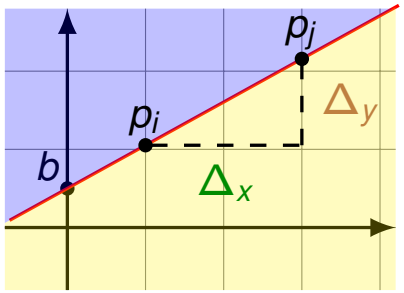


Esempi di applicazione

Inviluppo convesso: algoritmo semplice

- Data la retta $y = ax + b$,
  i due semipiani sono individuati da

$$y \leq ax + b \text{ e } y \geq ax + b$$



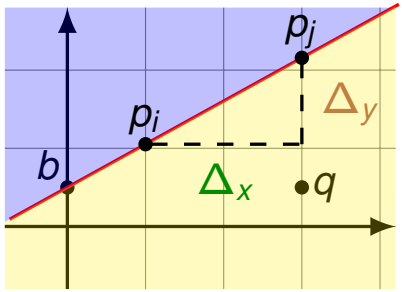
- Dati p_i, p_j della retta, $\Delta_x = p_{j_x} - p_{i_x}$, $\Delta_y = p_{j_y} - p_{i_y}$.
- $a \stackrel{\text{def}}{=} \Delta_y / \Delta_x \implies$ è possibile esprimere la retta come $y\Delta_x - x\Delta_y - b\Delta_x = 0$. Così si comprendono anche le rette "verticali".
- b in funzione di uno dei punti: $b = \frac{p_{i_y}\Delta_x - p_{i_x}\Delta_y}{\Delta_x}$
- Data una retta e un suo punto p, i semipiani possono essere descritti come:

$$y\Delta_x - x\Delta_y - p_y\Delta_x + p_x\Delta_y \stackrel{\leq}{\geq} 0$$

Esempi di applicazione

Inviluppo convesso: algoritmo semplice

- Dato un punto q generico e una delle due disequazioni precedenti (quella $\leq$ per esempio), sostituendo le coordinate si ottiene:



$$(q_y - p_y)\Delta_X - (q_x - p_x)\Delta_y \leq 0 \tag{1}$$

- Il punto q appartiene al semipiano "inferiore" se la disequazione è soddisfatta, altrimenti appartiene all'altro semipiano.
- I punti che stanno sulla retta appartengono ad entrambi i semipiani.

Esempi di applicazione

Inviluppo convesso: algoritmo semplice

- Dati due punti q, q' e una retta r, i due punti stanno dalla stessa parte se il prodotto

$$((q_y - p_y)\Delta_X - (q_x - p_x)\Delta_y)((q'_y - p_y)\Delta_X - (q'_x - p_x)\Delta_y) \geq 0$$

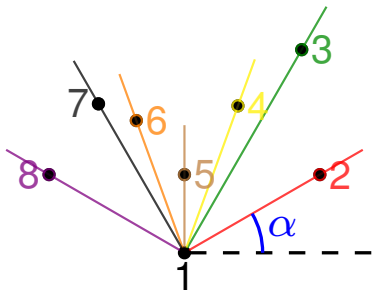
- Il costo di questa operazione è $O(1)$.
- Data una coppia (un potenziale spigolo) e scelto un punto a caso P, in $n - 3$ verifiche della disequazione si verifica se tutti i punti rimanenti stanno dalla stessa parte di P e quindi se la coppia individua uno spigolo.
- Ci sono $\frac{n(n-1)}{2}$ possibili coppie $\Rightarrow$ costo totale $O(n^3)$.

Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

Graham (1972) ha formulato un algoritmo più efficiente basato sull'ordinamento e un'analisi backtrack dei punti.

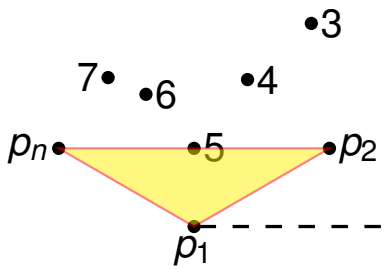
Osservazioni



- Il punto con ordinata minima è sempre un vertice di un inviluppo. Se ce ne sono più di uno, quelli con ascissa max e min formano due vertici: si sceglie quello di ascissa max.
- Se si traccia una semiretta da tale punto e la si ruota in senso antiorario a partire dall'asse x, si incontrano tutti i rimanenti punti.
- Si possono quindi numerarli in modo crescente in base all'angolo α. Equivale ad ordinarli.
- Se ci sono punti allineati $(= \alpha)$, si considera solo il punto più distante, eliminando gli altri. Quanto costa determinare tale ordinamento?

Esempi di applicazione

Inviluppo convesso: algoritmo di Graham



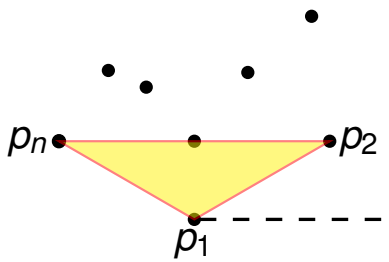
- Una volta ordinati, i primi due punti p_1 e p_2 e l'ultimo p_n sono vertici dell'inviluppo finale.
- Quindi p_n, p_1, p_2 costituiscono il punto di partenza della costruzione dell'inviluppo.
- Si estende l'inviluppo aggiungendo, uno alla volta, i rimanenti punti secondo l'ordine.
- Per ogni nuovo punto p_i aggiunto, si riesaminano tutti i punti che precedentemente sono stati aggiunti e si verifica se fanno ancora parte dell'inviluppo.
- Se risulta che un punto non può più far parte dell'inviluppo, si elimina (backtrack).

Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_j è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

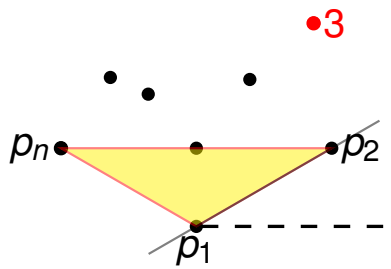


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

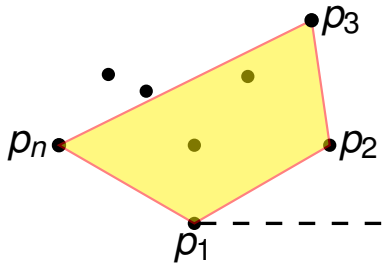


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

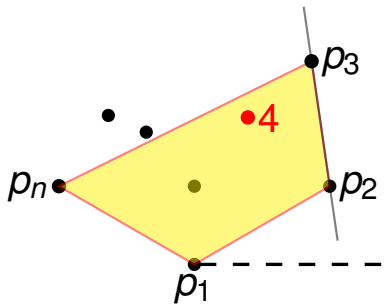


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

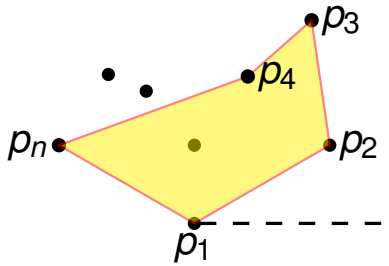


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

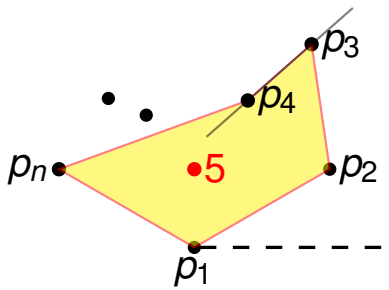


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

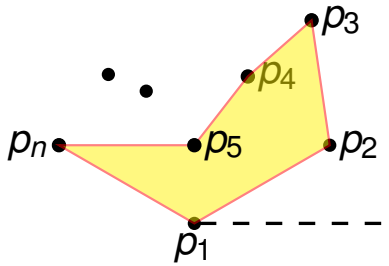


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

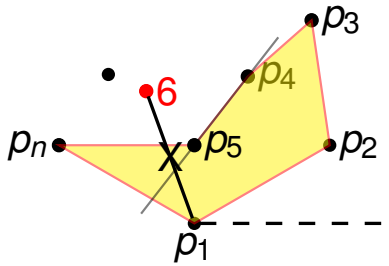


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

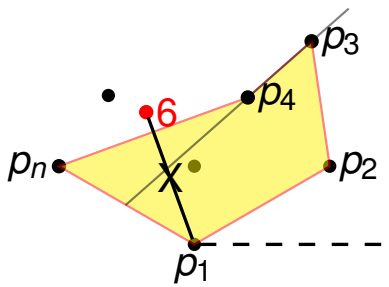


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

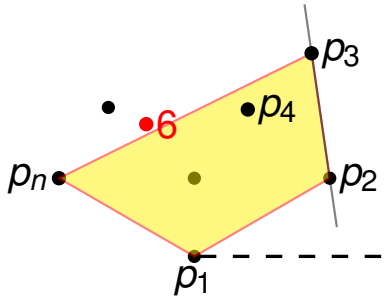


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

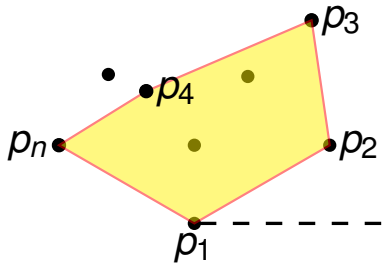


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.

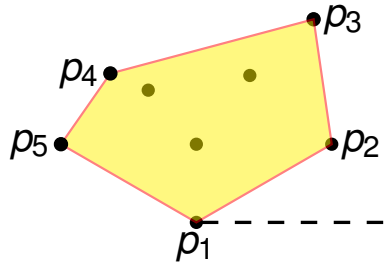


Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

In dettaglio:

- Dato
  un nuovo punto candidato p_i, $i = 3, \ldots, n$
- per ciascun $j = i - 1, \ldots, 2$:
  - Si disegna la retta passante per p_j e p_{j-1}.
  - Si verifica se i punti p_i e p_1 stanno nello stesso semipiano.
  - Se la verifica è positiva, allora si termina il ciclo e il punto p_i è parte dell'inviluppo.
  - Altrimenti, viene cancellato il punto p_j e i punti dell'inviluppo rinumerati.



Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

Procedura `Graham(`p`)`

// p è un vettore di punti, $p[i].x$ e $p[i].y$ = coordinate di p_i

1: $n = |p|$; $min = 1$;
2: **for** $(i = 2; \quad i \leq n; \quad i{+}{+})$ **do** // ricerca punto minimo
3: **if** $(p[i].y < p[min].y)$ **then** $min = i$;
4: **endfor**
5: $T = p[1]$; $p[1] = p[min]$; $p[min] = T$; // $p[1]$ è il punto minimo
6: Ordina $p[2], \ldots, p[n]$ in base all'angolo α;
7: Elimina eventuali punti allineati e aggiorna n;
8: $j = (n \geq 2)?2 : 1$; // $j =$ ultimo punto corretto dell'inviluppo

Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

continua...

```
8:  for (i = 3; i ≤ n; i++) do
9:      retta.p1 = p[j]; retta.p2 = p[j − 1];                    // retta per test
10:     while (!stessaParte(retta, p[1], p[i])) do
11:         j − −;                                  // punto j è interno, si elimina!
12:         retta.p1 = p[j]; retta.p2 = p[j − 1];
13:     endw
14:     j++;                          // = prima posizione utile per nuovo punto p_i
15:     T = p[j]; p[j] = p[i]; p[i] = T;                                // Sposta p_i
16: endfor
17: return p[1], . . . , p[j]
```

L'implementazione di Ordina, Elimina e stessaParte lasciata per esercizio.

Esempi di applicazione

Inviluppo convesso: algoritmo di Graham

Correttezza

La dimostrazione di correttezza è per induzione su i del ciclo **for**.

Complessità

- La fase di calcolo minimo costa $O(n)$.
- La fase di ordinamento costa $O(n \log n)$.
- Ci sono $n - 3$ cicli **for** ciascuno dei quali, apparentemente, sembra costare $O(n)$ per la presenza del **while**;
- In realtà, ogni ciclo **while** o termina subito, o elimina dei nodi e termina. Questi nodi eliminati non vengono più considerati. Quindi, nel complesso, tutti i cicli **while** costano $O(n)$.
- Nel complesso quindi il ciclo **for** costa $O(n)$.
- Il costo totale è quindi $O(n \log n)$.

Esempi di applicazione
RICERCA DI STRINGA

PROBLEMA RICERCA DI STRINGA (STRING MATCHING)

DESCRIZIONE: Una stringa T, detta testo, di dimensione n e una stringa P, detta anche pattern, di $m < n$ caratteri.

QUESITO: Determinare se esiste un'occorrenza di P in T.

In altre parole, si chiede se $\exists k,\ 1 \leq k \leq n - m$, tale che $T_{k+j-1} = P_j$ per $j = 1, 2, \ldots, m$.

Esempio 2

Se $T = 10110010101101011011011$ e $P = 10110110$, alla posizione 14 di T esiste un'occorrenza di P:

$$T = 10110010101101011011011$$

Esempi di applicazione

RICERCA DI STRINGA: algoritmo semplice

Soluzione semplice consiste nel cercare P ad ogni posizione di T.

Procedura `PatternMatchingForzaBruta(`T, P`)`

```
1:  n = |T|; m = |P|;
2:  i = j = k = 1;                          // k = indice inizio sottostringa in T
3:  while ((i + (m − j)) ≤ n && j ≤ m) do
4:      if (T[i] == P[j]) then
5:          i++; j++;
6:      else
7:          k++; i = k; j = 1;
8:      endif
9:  endw
10: if (j > m) then return k else return falso
```

Esempi di applicazione

RICERCA DI STRINGA: algoritmo semplice

Complessità

Nel caso peggiore, quando non esiste il pattern, per ogni posizione del testo si devono confrontare tutti i caratteri del pattern:
$O((n-m+1)m) = O(nm)$.

Esempi di applicazione

RICERCA DI STRINGA: considerazioni

- Il backtracking forza bruta non sfrutta il fatto di aver riconosciuto qualche carattere del pattern: se non trova il pattern da i, riparte da $i + 1$ ignorando ciò che ha appena letto.
- Si potrebbero sfruttare meglio le letture fatte per ripartire da una posizione $i + d$ per un qualche $d < m$.

Esempio 3

$T = 10110\,01010\ldots$ e $P = 10110\,110$.

Quando $i = j = 6 \Rightarrow$ backtrack $\Rightarrow i = 2, j = 1 \Rightarrow$

```
1011001010...
 10110110
```

Studiando i primi 5 caratteri del pattern...

```
10110
 10110 NO
```

```
10110
  10110 NO
```

```
10110
    10110 SI
```

Se è noto a priori che nei primi 5 caratteri di P solo gli ultimi 2 coincidono con i primi 2, si può ripartire da $i = k + (5 - 2) = 4$ e $j = 1$.

Esempi di applicazione

RICERCA DI STRINGA: considerazioni

continua...

Quindi si può ripartire come:

```
1011001010...
  10110110
```

La stringa 10 è già nota, $\Rightarrow$

```
1011001010...
  10110110
```

Alla fine, $i = 6$ e $j = 1 + 2 = 3 \Rightarrow$ si effettua il backtrack solo su j.

Occorre quindi conoscere, per ogni r, numero caratteri riconosciuti, qual è lo spostamento massimo a dx in P che si può effettuare prima di ricominciare il confronto.

Equivale a chiedere di determinare il più lungo suffisso proprio del pattern troncato al r° carattere.
L'implementazione di questo tipo di backtracking è realizzato dall'algoritmo di Knuth, Morris e Pratt (1977).

Esempi di applicazione

RICERCA DI STRINGA: algoritmo KMP

Definizione 1 (suffisso)

Date due stringhe P e S, di dimensione n e m rispettivamente, S è suffisso (proprio) di P se $m \leq n\ (m < n) \wedge P[n-i] == S[m-i]$ per $i = 0, \ldots, m-1$.

Definizione 2 (valore di backtrack)

Il valore di backtrack per l'indice j ($j-1$ caratteri riconosciuti) è definito come

$$back[j] = \max\{h+1 \mid h < j \wedge P_h \text{ è suffisso (proprio) di } P_j\}$$

con $j = 2, \ldots, m$ e dove P_h è la stringa P troncata al h-esimo carattere incluso.
Per convenienza, $back[1] = 0$.

Esempi di applicazione

RICERCA DI STRINGA: algoritmo KMP

Regola pratica per trovare $back[j]$:

- Si prendono due 2 copie del pattern troncato al $j-1$-esimo carattere;
- Si allinea la seconda copia sotto il 2° carattere della prima copia;
- Si verifica se i caratteri allineati sono uguali e, in caso negativo, si sposta a dx di 1 posizione la seconda stringa;
- Si termina quando o tutti i caratteri allineati sono uguali o le stringhe non sono sovrapposte.
- $back[j] =$ #caratteri allineati $+1$.

Esempi di applicazione

RICERCA DI STRINGA: algoritmo KMP

Esempio 4

$P = 10110110, |P| = 8 \Rightarrow j \leq 7.$

$P_2 = 10$, $back[3] = 1$ dato che

$$\begin{array}{l} 10 \\ \quad 10 \end{array}$$

$P_3 = 101$, $back[4] = 2$ dato che

$$\begin{array}{l} 10\underline{1} \\ \quad \underline{1}01 \end{array}$$

...

$P_7 = 1011011$, $back[8] = 5$ dato che

$$\begin{array}{l} 101\underline{1011} \\ \quad \underline{1011}011 \end{array}$$

Uso di *back*

Se durante il processo di riconoscimento, si verifica che $P[j] \neq T[i]$ per qualche $j < m$ e $i < n$, allora si aggiorna solo $j = back[j]$.

Esempi di applicazione

RICERCA DI STRINGA: algoritmo KMP

Algoritmo di *Knuth, Morris e Pratt*

Procedura KMP(T, P)

```
1:  n = |T|; m = |P|;
2:  back =calcolaBack(P);
3:  i = j = 1;
4:  while ((i + (m − j)) ≤ n  &&  j ≤ m) do
5:      if (j == 0 || T[i] == P[j]) then // j == 0 ⇔ 0 caratteri riconosciuti
6:          i++; j++;
7:      else
8:          j = back[j];
9:      endif
10: endw
11: if (j > m) then return i − m else return non esiste
```

Esempi di applicazione

RICERCA DI STRINGA: algoritmo KMP

Per `calcolaBack`(P) si può riapplicare lo schema dell'algoritmo `KMP()` su P_i

Funzione `calcolaBack`(P)

```
1:  back[1] = 0; back[2] = 1;                          // Caso base
2:  j = 2; h = 1;                          // j = indice stringa, h = indice suffisso
3:  while (j < m) do
4:      if (h == 0 || P[j] == P[h]) then
5:          j++; h++;
6:          back[j] = h;                   // back[j] = #caratteri ric. di P_{j-1} + 1
7:      else
8:          h = back[h];                   // back[h] = #caratteri ric. fino a h.
9:      endif
10: endw
11: return back
```

Esempi di applicazione

RICERCA DI STRINGA: algoritmo KMP

Esempio 5 (Esempio di calcolo *back*)

$P =$	10110110		$back[1] \stackrel{\text{def}}{=} 0$
			$back[2] \stackrel{\text{def}}{=} 1$
$P =$	10110110	$j = 2, h = 1$	
	1	$j = 2, h = back[1] = 0$	
		$j = 3, h = 1$	$back[3] = 1$
$P =$	10110110	$j = 3, h = 1$	
	1	$j = 4, h = 2$	$back[4] = 2$
	10	$j = 4, h = back[2] = 1$	
$P =$	10110110	$j = 4, h = 1$	
	1	$j = 5, h = 2$	$back[5] = 2$
	10	$j = 6, h = 3$	$back[6] = 3$
	101	$j = 7, h = 4$	$back[7] = 4$
	1011	$j = 8, h = 5$	$back[8] = 5$

Esempi di applicazione

RICERCA DI STRINGA: algoritmo KMP

Ulteriore ottimizzazione di calcolaBack

- In `KMP()`, $j = back[j]$ quando $T[i] != P[j]$.
- In `calcolaBack()`, $back[j] = h$ quando $P[j] == P[h]$ o $h == 0$.
- Quindi, in `KMP()`, quando $T[i] != P[j]$, allora $T[i] != P[h]$ se $P[j] == P[h]$ e quindi si deve fare un secondo "backtrack" .
- Per evitare, è sufficiente "assorbire" il secondo salto già in `calcolaBack()`: si sostituisce back[j]=h con
  if $(P[j] == P[h])$ **then** back[j]=back[h]; **else** back[j]=h;
- In questo modo in `KMP()` l'indice j è decrementato al più 1 volta per ogni valore di i.

Esempi di applicazione

RICERCA DI STRINGA: algoritmo KMP

Complessità

In `KMP()` il ciclo **while** è eseguito n volte al più e, per quanto detto sopra, ci sono $O(1)$ operazioni per ciclo.
La funzione `calcolaBack()` richiede tempo $O(m)$.
L'algoritmo quindi ha complessità $O(n + m)$.

Altri problemi risolvibili con backtracking

Elenco incompleto di problemi che ammettono un buon algoritmo di risoluzione del tipo backtracking.

Problema	**Costo naïve**	**Costo backtracking**
Il problema delle 8 regine	$O(8!)$	$O(114)$
...		

Esercizi

Problema dello ZAINO

Esercizio 3

Si consideri la seguente istanza del problema zaino:
$v = [25, 8, 19, 26, 10, 10]$, $w = [6, 2, 5, 7, 3, 4]$ e $P = 12$.
Si descriva un algoritmo backtracking per determinare la soluzione ottima e si scriva l'albero implicito visitato.

Esercizi

MINIMO RICOPRIMENTO DI VERTICI

PROBLEMA MINIMO RICOPRIMENTO DI VERTICI (MINIMUM VERTEX COVER)

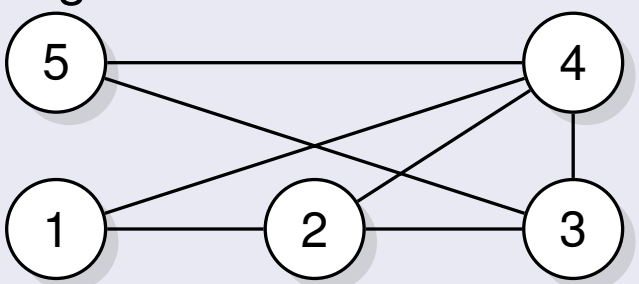
DESCRIZIONE: Un grafo $G = (V, E)$.

QUESITO: Determinare un sottoinsieme di vertici $U \subseteq V$ tale che $\forall \{v_i, v_j\} \in E \Rightarrow v_i \in U \vee v_j \in U$.

MISURA: |U|

Esercizio 4

- Si determini un algoritmo backtracking per il problema.
- Si scriva l'albero implicito determinato dall'algoritmo quando applicato al seguente grafo:



Esercizi

Tutte le permutazioni

Esercizio 5

Si scriva un algoritmo basato sul backtracking che stampi tutte le permutazioni di n caratteri di una stringa S.

Suggerimento: Una permutazione di n caratteri può essere vista come una permutazione di $n - 1$ *seguita dal carattere mancante.*