Problemi difficili e ricerca esaustiva intelligente

Progettazione di Algoritmi a.a. 2021-22

Matricole congrue a 1

Docente: Annalisa De Bonis

1

- Gli argomenti di questa lezione sono tratti da Dasgupta, Papadimitriou, Vazirani: Algorithms : Cap. 9, fino al paragrafo 9.1.2 (incluso)
- Nella lezione vengono inoltre illustrati, in modo informale, alcuni concetti di teoria della complessità richiamati nell'introduzione al capitolo.

2

Introduzione

Finora ci siamo interessati alla progettazione di algoritmi efficienti

Efficienza = polinomiale

Tecniche: divide et impera, greedy, programmazione dinamica

Però….. esistono dei problemi

- Che non sappiamo risolvere efficientemente
- Né sappiamo dimostrare che non esistono soluzioni efficienti

P e NP

Problema decisionale: consiste nel rispondere ad una domanda che ammette solo sì o no come risposta

Informalmente:

P è la classe dei problemi decisionali risolvibili in tempo polinomiale

Es.: esiste un cammino di costo al più k fra due vertici?

Usa Dijkstra per trovare il cammino minimo e se questo ha costo al più k rispondi sì; altrimenti rispondi no.

NP è la classe dei problemi decisionali verificabili in tempo polinomiale,

informalmente, problemi per cui è possibile esibire una prova che esiste una soluzione al problema (risposta al problema decisionale è si`) ed è possibile verificare in tempo polinomiale che essa è una prova dell'esistenza di una soluzione al problema

Es.: cammino hamiltoniano: Dato un grafo, determinare se esiste un percorso che tocca i vertici del grafo esattamente un volta.

Se ci viene fornita in input una sequenza di vertici possiamo in tempo polinomiale verificare che si tratta di un percorso che tocca ciascun vertice del grafo esattamente una volta

P = NP?

Chiaramente P $\subseteq$ NP.

Ma P = NP? Oppure P≠NP? Non si conosce la risposta!

P = NP implicherebbe che ogni problema in NP può essere risolto in tempo polinomiale.

per chi risolvesse il problema P=NP? c'è in palio…

Clay Mathematics Institute

Problemi NP-completi

- I problemi NP-completi sono problemi in NP difficili **almeno** quanto qualsiasi altro problema in NP
  - Se si trovasse un algoritmo polinomiale per uno solo di questi problemi allora qualsiasi altro problema in NP risulterebbe risolvibile in tempo polinomiale (P=NP)
- Il problema del cammino Hamiltoniano è NP-completo
  - Se si trovasse un algoritmo polinomiale per questo problema allora ogni problema in NP potrebbe essere risolto in tempo polinomiale

Problemi NP-completi

- Se qualcuno fosse in grado di trovare un algoritmo polinomiale per uno qualsiasi dei problemi NP-completi, questa persona vincerebbe

- Pochi credono che ciò sia possibile… La maggior parte degli esperti di teoria della complessità computazionale propende per l'ipotesi P≠NP

Problemi non decisionali

- Se un problema non è un problema decisionale possiamo comunque classificarlo secondo le modalità viste prima tra i problemi risolvibili in modo efficiente o in quelli difficili
- Problema NP-hard: problema (non necessariamente in NP) la cui soluzione in tempo polinomiale permetterebbe la soluzione in tempo polinomiale di ogni problema in NP.

Problemi di ottimizzazione

- Un modo per classificare un problema di ottimizzazione come problema difficile è di formularlo come un problema decisionale
- Se la versione decisionale è NP-completa allora il problema è NP-hard
- Esempio: Problema del commesso viaggiatore
  - Dato un grafo non direzionato con dei costi non negativi sugli archi, trovare un ciclo di costo minimo che passa esattamente una volta attraverso ciascun nodo
  - Possiamo riformularlo come "Esiste un ciclo che passa esattamente una volta attraverso ciascun nodo e ha costo al più k?"
  - E' evidente che il problema di ottimizzazione non è più semplice da risolvere della sua versione decisionale. Se conosco la soluzione del problema di ottimizzazione posso rispondere immediatamente alla domanda del problema decisionale.

Il problema della fattorizzazione di un intero

•La difficoltà di alcuni problemi non è solo una questione teorica e non deve essere in generale vista come una cosa negativa.

•Problema della fattorizzazione di un intero in una delle sue versioni più note: dato un intero positivo $n = a \cdot b$, con a e b primi, trovare a e b.

• Se a e b sono primi molto grandi è molto difficile risolverlo, ma se qualcuno mi suggerisce a e b, posso facilmente verificare che $n = a \cdot b$

•Il problema della fattorizzazione di interi è usato nel sistema crittografico RSA che è alla base del commercio elettronico!

•se si trovasse un algoritmo efficiente per fattorizzare un intero allora non potremmo più fare affidamento su RSA.

N.B. non e` noto se questo problema è NP-hard.

Ricerca esaustiva

Supponiamo di dovere fornire un algoritmo per risolvere un problema.

Se non riusciamo a trovare un algoritmo efficiente con nessuna tecnica studiata, potrebbe trattarsi di un problema “difficile”!

Che fare?

Unica possibilità: la forza bruta!

Ricerca esaustiva?

Ricerca esaustiva su tutto lo spazio delle soluzioni.

In linea di principio risolve qualsiasi problema!

Ma in quanto tempo?

Per la fattorizzazione di n, se a e b avessero 2048 bit, non basterebbe una vita!

Tuttavia la tecnica può essere utilizzata per istanze piccole.

Vedremo adesso degli accorgimenti per migliorare l’efficienza di algoritmi di ricerca esaustiva

Ricerca esaustiva intelligente

Backtracking: a volte si può rifiutare una possibile soluzione esaminando solo una sua piccola parte

Branch and bound: in un problema di ottimizzazione, a volte si può scartare una possibile soluzione senza averla esaminata completamente, perché il suo valore non può essere ottimale

Formule booleane e soddisfacibilità

Una formula (espressione) booleana ϕ è soddisfacibile se esiste un assegnamento delle sue variabili che la rende Vera/ 1

Es.: $\phi = (a + \neg b)\cdot(\neg a)$ è soddisfacibile perchè $\phi = 1$ assegnando a=0, b= 0
$\phi = (a)\cdot(\neg a)$ non è soddisfacibile

Stabilire se una formula booleana è soddisfacibile è «difficile» (problema NP-Completo)

Ogni formula booleana può essere espressa in Forma Normale Congiuntiva, in breve CNF (espressione POS).

Il problema SAT

SAT (Satisfiability)

INPUT: una formula booleana

OUTPUT: SI, se la formula è soddisfacibile; NO, altrimenti

CNF - SAT

INPUT: una formula booleana in CNF

OUTPUT: SI, se la formula è soddisfacibile; NO, altrimenti

SAT e CNF-SAT sono due problemi decisionali NP-completi. Qui ci occuperemo di CNF-SAT

Risolvere SAT

Esempio: $\phi = (a + b + c + \neg d) \cdot (a + b) \cdot (a + \neg b) \cdot (\neg a + c) \cdot (\neg a + \neg c)$

Per vedere se ϕ è soddisfacibile posso valutare tutti i 16 assegnamenti possibili
Dato uno specifico assegnamento, è facile verificare se esso rende ϕ Vera/1 o Falsa/0.

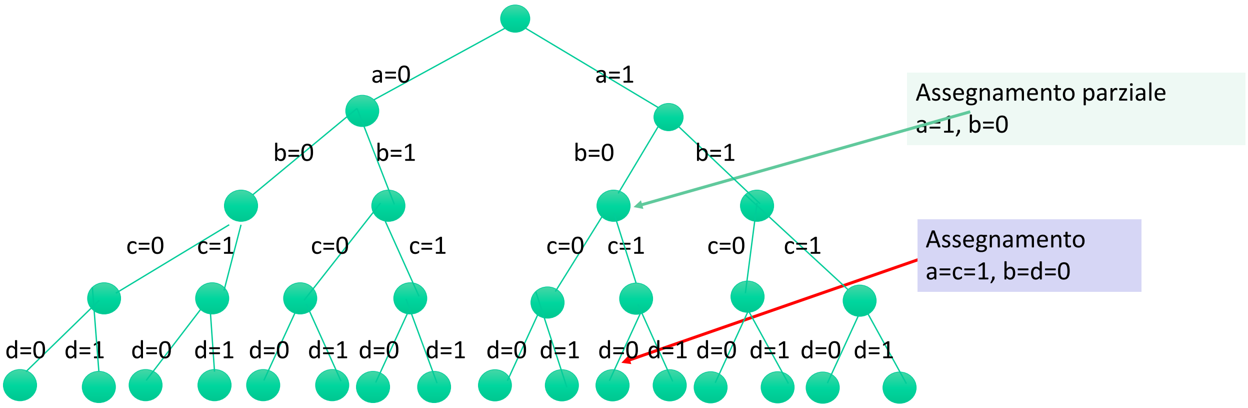
Per esempio: per a=1, b=0, c=1, d=0 si ha

$\phi = (1 + 0 + 1 + \neg 0) \cdot (1+0) \cdot (1 + \neg 0) \cdot (\neg 1 + 1) \cdot (\neg 1 + \neg 1) =$

$= 1 \cdot 1 \cdot 1 \cdot 1 \cdot 0 = 0$

Ricerca esaustiva e albero delle decisioni

Potremmo organizzare tale processo tramite un albero delle decisioni



L'albero avrà 2^n foglie, se n è il numero delle variabili. Dovremmo valutare un numero esponenziale di possibili assegnamenti.

17

Backtracking

Cominciamo dalla radice costruendo l'albero in maniera incrementale.

A volte potremo scartare alcuni assegnamenti, fermandoci ad un assegnamento parziale, dal quale è inutile proseguire.

Allora backtrack: torniamo indietro per provare altre strade.

18

Backtracking e SAT

Esempio: $\phi(a,b,c,d) = (a + b + c + \neg d) \cdot (a + b) \cdot (a + \neg b) \cdot (\neg a + c) \cdot (\neg a + \neg c)$

Partiamo ponendo a=0

$\phi(0, b,c,d) = (0 + b + c + \neg d) \cdot (b) \cdot (\neg b) \cdot (1) \cdot (1) = (b + c + \neg d) \cdot (b) \cdot (\neg b)$

Sottoproblema: $\phi(b,c,d) = (b + c + \neg d) \cdot (b) \cdot (\neg b)$ è soddisfacibile?

Poniamo b=0.

$\phi(0,0,c,d) = (c + \neg d) \cdot (0) \cdot (1) = 0$

indipendentemente dagli assegnamenti per c e d: Backtrack!

19

Valutazione parziale

Nell'albero delle decisioni:

$$\phi(a,b,c,d) = (a + b + c + \neg d) \cdot (a + b) \cdot (a + \neg b) \cdot (\neg a + c) \cdot (\neg a + \neg c)$$

a=0

$\phi(0,b,c,d) = (0 + b + c + \neg d) \cdot (0+b) \cdot (0+\neg b) \cdot (1+c) \cdot (1+ \neg c)$

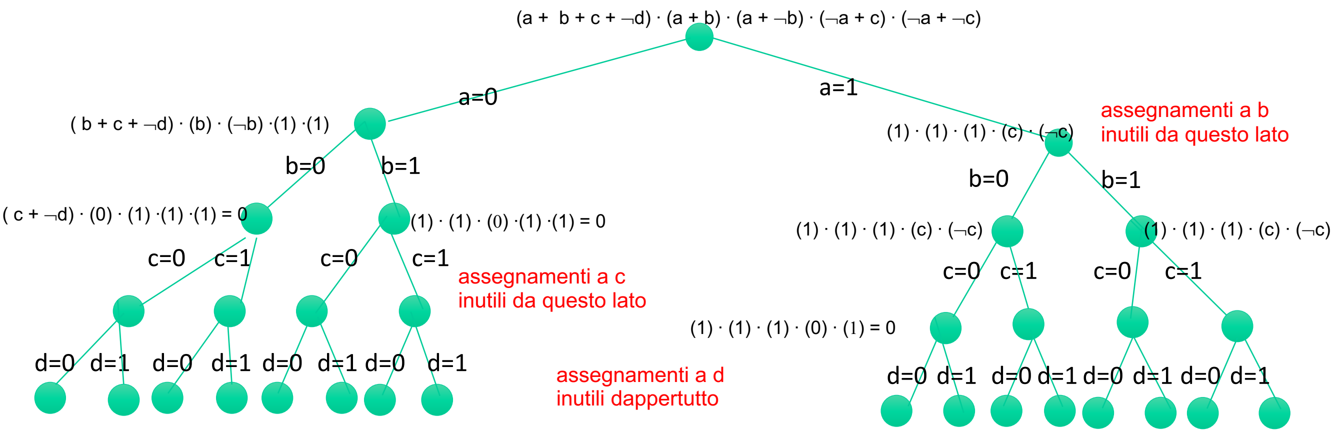
$= (b + c + \neg d) \cdot (b) \cdot (\neg b)$

b=0

$\phi(0,0,c,d) = (0 + c + \neg d) \cdot (0) \cdot (1) = 0$

20

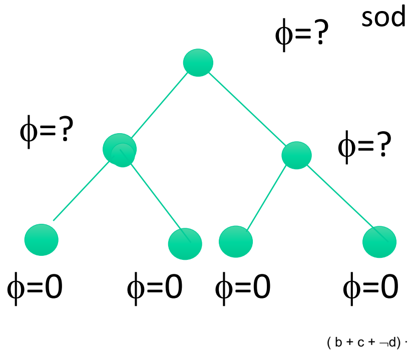
Possiamo fermarci dopo aver considerato 4 assegnamenti parziali



PROGETTAZIONE DI ALGORITMI A.A. 2021-22
A.DE BONIS

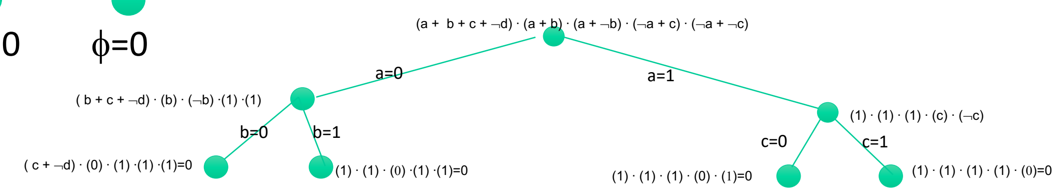
21

Alla fine....



Possiamo affermare che la formula non è soddisfacibile.

Avendo risparmiato un bel po' di calcoli



PROGETTAZIONE DI ALGORITMI A.A. 2021-22
A.DE BONIS

22

Backtracking

Per ogni sottoproblema considerato esegue un test con 3 possibili risultati:

Failure: il sottoproblema non ha soluzioni
(es.: ϕ = 0 in un nodo interno)

Success: trovo una soluzione al problema di partenza
(es.: ϕ =1 in una foglia)

Uncertainty: bisogna proseguire
(es.: ϕ =?)

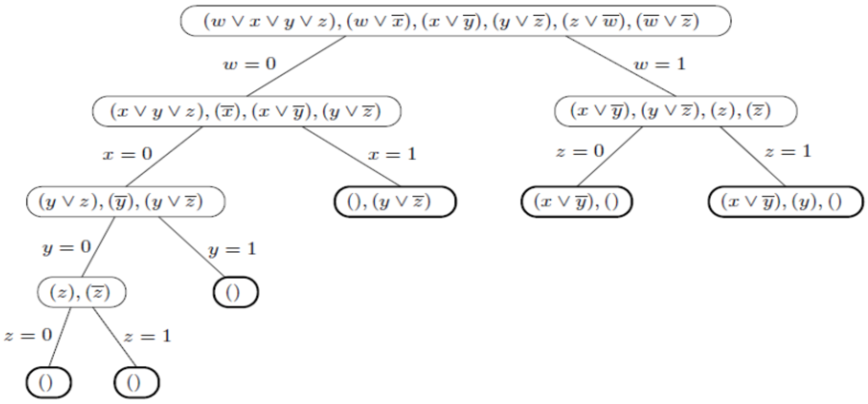
Schema backtrack

```
Start with some problem P_0
Let S = {P_0}, the set of active subproblems
Repeat while S is nonempty:
  choose a subproblem P ∈ S and remove it from S
  expand it into smaller subproblems P_1, P_2, ..., P_k
  For each P_i:
    If test(P_i) succeeds:  halt and announce this solution
    If test(P_i) fails:  discard P_i
    Otherwise:  add P_i to S
Announce that there is no solution
```

Nonostante la complessità resti esponenziale, il backtracking può essere molto efficiente nella pratica

Esempio del libro

Figure 9.1 Backtracking reveals that ϕ is not satisfiable.



Qui () indica una «clausola vuota» cioè insoddisfacibilità

Branch and bound

La stessa idea del backtracking può essere estesa a problemi di ottimizzazione.

Ogni soluzione ha un valore e cerchiamo una soluzione ottima (minima o massima). Nel seguito considereremo problemi di minimizzazione.

Anche stavolta considereremo soluzioni parziali a sottoproblemi.

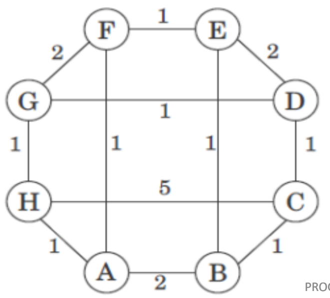
Per escludere una soluzione parziale dobbiamo essere certi che il suo costo supera quello di un'altra soluzione già calcolata. Meglio: il costo di tutte le soluzioni sviluppate a partire da essa hanno un costo superiore ad un certo limite (bound).

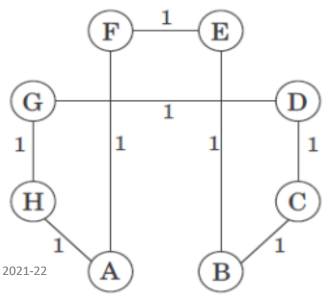
Il commesso viaggiatore

Il problema del commesso viaggiatore, in breve TSP (Traveling Salesman Problem) è il seguente.

TSP
INPUT: Un grafo completo G=(V,E) con dei costi sugli archi
OUTPUT: Un ciclo di costo minimo che passa per ogni vertice una ed una sola volta (giro di costo minimo)

nel grafo sono stati rimossi gli archi con peso molto grande





27

Risolvere TSP

La ricerca esaustiva valuta ogni possibile giro del grafo a partire da un certo nodo (non ha importanza quale)

In genere si assume che il grafo è completo (un arco tra ciascuna coppia di nodi) per cui occorre considerare tutte le sequenze di n+1 vertici del grafo tale che

- La sequenza comincia e finisce in uno stesso vertice (non ha importanza quale)
- I nodi compresi tra il primo e l'ultimo compaiono esattamente una volta nella sequenza

Numero di sequenze = $(n-1)!$

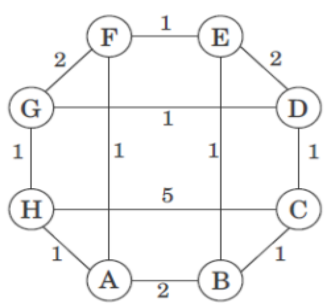
28

Risolvere TSP: albero delle decisioni

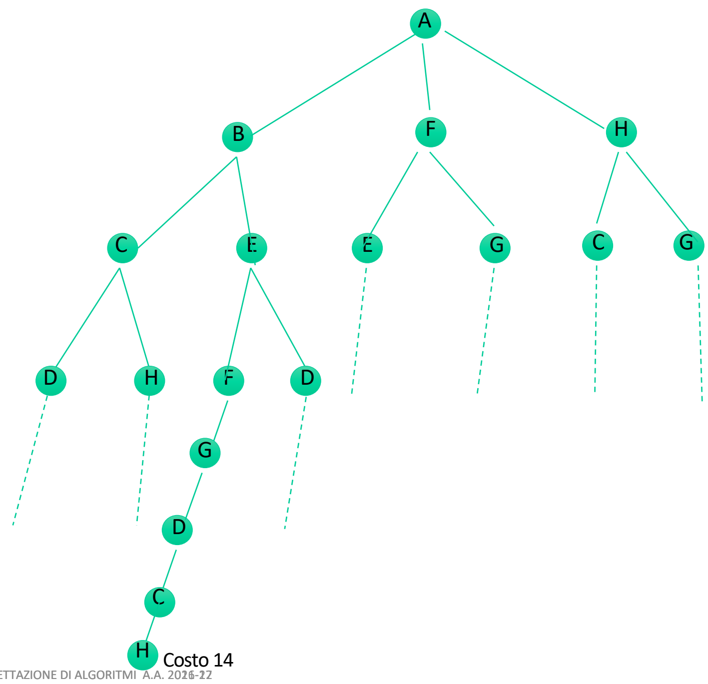
- Il nodo prescelto come punto di partenza diventa radice dell'albero delle decisioni.
- Ogni foglia rappresenta un giro.
- Ogni nodo interno un cammino parziale dal nodo di partenza.
  - Un nodo interno v ha come figli i nodi ad esso adiacenti che non sono stati ancora attraversati dal **cammino parziale** che termina in v.
- N.B. Il numero di foglie è esponenziale nel numero di nodi del grafo

29

Albero delle decisioni



N.B. Il grafo input non è completo: si assume che siano stati tolti gli archi più costosi che non possono far parte della soluzione ottima.



30

Risolvere TSP con Branch and Bound

•Partendo dal nodo prescelto come punto di partenza dall'algoritmo, costruiamo in modo incrementale l'albero delle decisioni.

- Il nodo prescelto diventa radice.
- Ogni foglia rappresenta un giro.
- Ogni nodo interno un cammino parziale dal nodo di partenza.

•Nel costruire l'albero delle decisioni in modo incrementale, ogni volta che arriviamo in un nodo interno valutiamo se estendere ulteriormente il cammino parziale che termina in quel nodo.

Limite inferiore

Per escludere una soluzione/cammino parziale bisogna essere sicuri che il costo di ogni suo completamento porti ad un costo maggiore di un certo limite inferiore.

Sia x il vertice di partenza (nell'esempio x=A)

Ogni nodo interno v dell'albero rappresenta un cammino P semplice da x a v che passa per un certo insieme di nodi **S che include anche** x e v. Indichiamo con [x, S, v] questo cammino parziale.

Vogliamo trovare un limite inferiore al costo di qualsiasi giro completo che inizia con P.

Se tale limite è ≥ del costo di un giro già noto, possiamo evitare di esplorare il sottoalbero di v.

Limite inferiore per TSP

Come limitare inferiormente il costo di un percorso che comincia con P?

Osserviamo che un tale percorso proseguirà da v attraversando solo nodi di V\S e poi raggiungerà di nuovo il punto di partenza x.

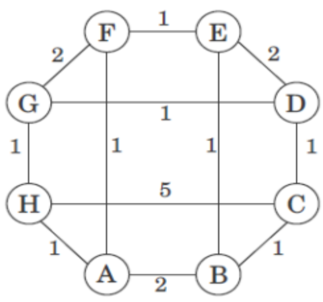
Il costo del percorso che resta ancora da fare è ≥ della somma di:

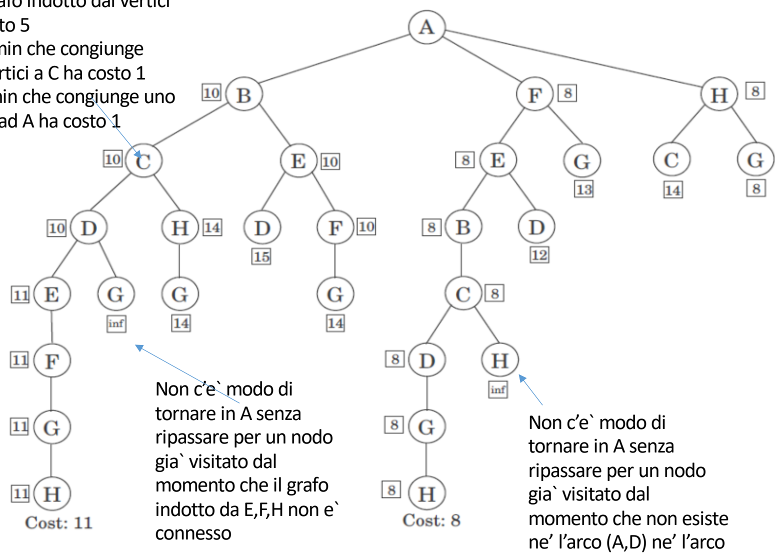
1. L'arco di costo minimo tra v ed un vertice in V\S
2. L'arco di costo minimo tra un vertice in V\S e x
3. Il costo di un MST del sottografo formato dai nodi in V\S e dagli archi di E che incidono su questi nodi (sottografo indotto dai nodi in V\S)
   - NB: un percorso semplice in V\S non può avere costo inferiore all'albero di costo minimo che congiunge tutti i vertici in V\S.

33

Branch and bound: esempio

MST del sottografo indotto dai vertici D,E,F,G,H ha costo 5
L'arco di costo min che congiunge uno di questi vertici a C ha costo 1
L'arco di costo min che congiunge uno di questi vertici ad A ha costo 1





Non c'e` modo di tornare in A senza ripassare per un nodo gia` visitato dal momento che il grafo indotto da E,F,H non e` connesso

Non c'e` modo di tornare in A senza ripassare per un nodo gia` visitato dal momento che non esiste ne' l'arco (A,D) ne' l'arco (A ,G)

I numeri nei quadrati sono il limite ottenuto sul giro che inizia in quel modo.

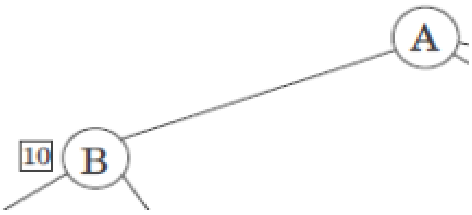
Alla fine considereremo soltanto 11 giri

34

Esempio

Consideriamo il cammino parziale P= (A, B)

Costo di P=2



Costo di una qualsiasi estensione di P ≥ 8

1. 1 = costo arco BC
2. 1 = costo arco AH
3. 6 costo MST relativo a {C, D, E, F, G, H}

L'arco AB ha costo 2 per cui il costo di un giro completo che inizia con P è ≥ 10

Schema algoritmo branch and bound

Start with some problem P_0
Let $\mathcal{S} = \{P_0\}$, the set of active subproblems
bestsofar $= \infty$
Repeat while $\mathcal{S}$ is nonempty:
  choose a subproblem (partial solution) $P \in \mathcal{S}$ and remove it from $\mathcal{S}$
  expand it into smaller subproblems $P_1, P_2, \ldots, P_k$
  For each P_i:
    If P_i is a complete solution: update bestsofar
    else if lowerbound$(P_i) <$ bestsofar: add P_i to $\mathcal{S}$
return bestsofar