PER ALTRI APPUNTI CONSULTARE IL SITO:

https://luigi-v.github.io/Appunti_Universita/

DISCLAIMER:

Sono esercizi svolti da me durante la preparazione dell’esame e non corretti dal prof, pertanto, alcuni esercizi possono essere sbagliati.

PAG: 48 | ESERCIZIO 3: CAP 1

TEORIA (PAG 12):

La difficoltà nella risoluzione di un problema è in funzione alla "taglia" dell'istanza da risolvere. Per misurare la grandezza dell'istanza del problema bisogna stabilire una codifica ragionevole del problema, ad esempio in termini di stringhe binarie, successivamente misurare la lunghezza di tale stringa.

(a) RAPPRESENTAZIONE BINARIA PER CLIQUE

Per specificare una istanza del problema CLIQUE codifichiamo il numero di nodi e archi con una stringa del tipo:

$$n, (v_1, v_2), \ldots (v_i, v_j) \ldots \#$$

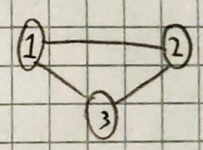
dove n è il numero di nodi, (v_i, v_j) sono le tuple che rappresentano gli archi. Codifichiamo ognuno di questi elementi con delle sequenze di bit, ottenendo una singola stringa binaria:

10 cifre - virgole - parentesi tonde - fine str

In totale servono 14 simboli, che possono essere rappresentati con 4 bit:

0000 = 0	1010 = ,
0001 = 1	1011 = (
⋮	1101 =)
1001 = 9	1111 = #

ESEMPIO ISTANZA:



3, (1,2), (2,3), (3,1)#

(b) PARAMETRO PIÙ SIGNIFICATIVO E RELAZIONE RISPETTO LA LUNGHEZZA DELLA STRINGA:

Il parametro più significativo è il numero di nodi (n), siccome esso determina il numero massimo di archi, ovvero abbiamo massimo n^2 archi.

(c) VERSIONE DECISIONALE DI CLIQUE / (d) ADATTARLO NELLA VER. DECISIONALE

Un qualsiasi problema di ottimizzazione possiamo trasformarlo in un problema decisionale, introducendo un parametro k e chiedendoci:

Esiste una CLIQUE di taglia k?

ESERCIZIO 4: CAP 1

TEORIA (PAG: 15):

Quando un problema non è in P, per mostrare che è in NP dovremmo formare un verificatore che si limiterà a controllare il certificato.
Per verificare una potenziale soluzione avremo bisogno di un certificato, cioè una codifica della potenziale soluzione da verificare.
Un verificatore V è efficiente per un problema X se:

- V è un algoritmo polinomiale che prende in input due stringhe, s che è l'input codificato in una stringa binaria del problema, e c che è il certificato
- Esiste una funzione $p()$ tale che per ogni stringa s, si ha che $s \in X$ se e solo se esiste una stringa c tale che $|c| \leq p(|s|)$ e $V(s,c) = sì$.

Richiede che V sia in grado di controllare che il certificato c rappresenti una soluzione all'istanza s del problema e deve essere in grado di farlo in tempo polinomiale.

ESERCIZIO, FORNIRE UN VERIFICATORE PER I PROBLEMI:

Il verificatore si limiterà a controllare il certificato

- TSP: basta verificare che un potenziale giro del grafo sia effettivamente un giro e abbia un costo $\leq$ alla soglia stabilita;
- CIR.SAT: basta verificare se un dato assegnamento dell'input produca il valore vero;
- SAT: basta verificare se, data una formula booleana, un dato assegnamento alle variabili rende vera la formula;
- IS: dato un insieme di nodi, basta verificare che forma un insieme di nodi indipendenti di taglia k.
- ... (continua a PAG: 15)

TEMPO NECESSARIO PER LA VERIFICA DELLA SOLUZIONE

Il parametro più significativo è legato polinomialmente alla lunghezza della stringa binaria che rappresenterebbe il problema, pertanto, anche la verifica di tale stringa dipende dal parametro più significativo.

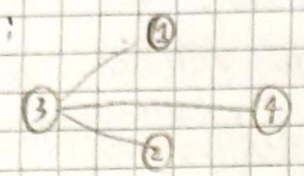
ESERCIZIO 6: CAP 1

TEORIA: (PAG 15)

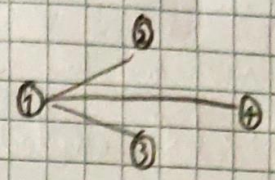
GRAFI ISOMORFI IN NP:

Dati due grafi:

G_1:



G_2:



La verifica che sono isomorfi è un problema in NP.

Per dimostrare ciò, bisogna esibire un algoritmo V in grado di controllare che il certificato c (codifica di una soluzione) rappresenti una soluzione all'istanza s del problema e deve farlo in tempo polinomiale.

$V(s,c)$

L'algoritmo prenderà due grafi ($\overset{s}{G_1}$ e $\overset{c}{G_2}$) e verifica se i grafi sono effettivamente isomorfi, cioè $V(s,c) = SI$. Tutto ciò fatto in tempo polinomiale per definizione di V.

ESERCIZIO 7: CAP 1

TEORIA (PAG: 16)

Un problema A può essere ridotto a B, se è possibile risolvere una qualsiasi istanza di A usando tempo polinomiale e sfruttando un oracolo che risolve B.

IN UNA RIDUZIONE, QUANTE VOLTE PUÒ ESSERE CHIAMATO L'oracolo?:

L'oracolo può essere chiamato più di una volta, siccome per definizione, l'oracolo è capace di risolvere qualsiasi problema in tempo costante, pertanto, se chiamato più volte il tempo complessivo rimane costante o al più polinomiale.

ESERCIZIO 1: CAP 1

TEORIA (PAG: 18/9)

VERTEX COVER $\leq_T$ INDIPENDENT SET

Q. DESCRIZIONE INDUZIONE:

Per provare la riduzione bisogna risolvere VC sfruttando l'oracolo che risolve IS usando tempo polinomiale.

1. CONSIDERARE ISTANZA VC (1° PROBLEMA):

Considerando un'istanza di VC, costituita da un grafo G e un intero k, dobbiamo decidere se G ha un insieme ricoprente di taglia al massimo k.

2. COSTRUIRE ISTANZA DI IS (2° PROBL.) IN RELAZIONE A VC (1 PROBL.) PER USARE L'ORACOLO:

Costruiamo un'istanza di IS, costituita da un grafo $G' = G$ e un intero $k' = n - k$ dove n è il numero di nodi. Questo passaggio richiede tempo polinomiale.
Una volta fatto ciò, possiamo dare l'istanza di IS all'oracolo in modo tale che la soluzione per IS fornita dall'oracolo trovi una soluzione per l'istanza di VC.
L'oracolo dirà se G' ha un insieme indipendente di taglia almeno k'.

3. CONSIDERARE RISP ORACOLO:

Sia $r \in \{SI, NO\}$ la risposta dell'oracolo, per il lemma 1.5.7 (PAG: 18) si ha che r è anche la risposta alla domanda "Esiste un insieme ricoprente di al massimo $k = n - k'$ nodi?", quindi r è anche la risposta all'istanza di VC.

ESERCIZIO 2: CAP 1

TRONA (PAG: 20/1)

INDIPENDENT SET $\leq_T$ SET PACKING

0. DESCRIZIONE DELLA RIDUZIONE:

Per provare la riduzione bisogna risolvere IS sfruttando l'oracolo che risolve SP usando tempo polinomiale.

1. CONSIDERARE ISTANZA IS (1° PROBL.):

Consideriamo un'istanza di IS, costituita da un grafo $G=(V,E)$ e un intero k, dobbiamo decidere se G ha un insieme indipendente di taglia almeno k.

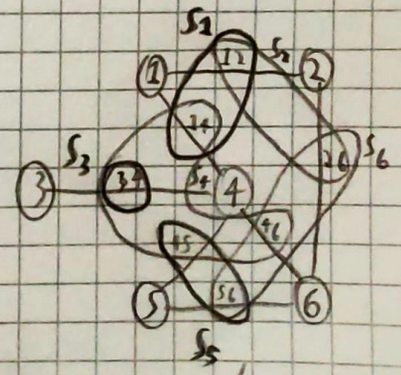
2. COSTRUIRE ISTANZA SP (2° PROBL) IN RELAZIONE A IS (1° PROBL) PER USARE L'ORACOLO:

Abbiamo a disposizione un oracolo per SP, per usarlo costruiamo una istanza per SP costituita da un insieme U, un sottoinsieme di U $(S_1, S_2, \dots S_m)$ e un intero k'.
L'insieme U è costituito da tutti gli archi di G, cioè $U=E$. Per ogni vertice $i \in V$, consideriamo l'insieme $S_i \subseteq U$, costituito da tutti gli archi di E incidenti su i.
L'istanza di SP ammette un insieme indipendente di almeno k' degli insiemi $S_1, \dots S_m$ se e solo se il grafo G ammette un insieme indipendente di almeno k nodi.

- Sia $S_{i_1}, \dots S_{i_k}$ un insieme indipendente per U, allora si ha che ogni vertice associato a $S_{i_1}, \dots S_{i_k}$ è un insieme indipendente di G di taglia $k \leq k$.
- Analogamente, sia $\{i_1, \dots i_k\}$ un insieme indipendente per G, allora $S_{i_1}, \dots S_{i_k}$ è un insieme indipendente per U.

3. CONSIDERANDO RISP ORACOLO:

Considerando la risposta dell'oracolo per l'istanza di SP è anche la risposta al problema IS.

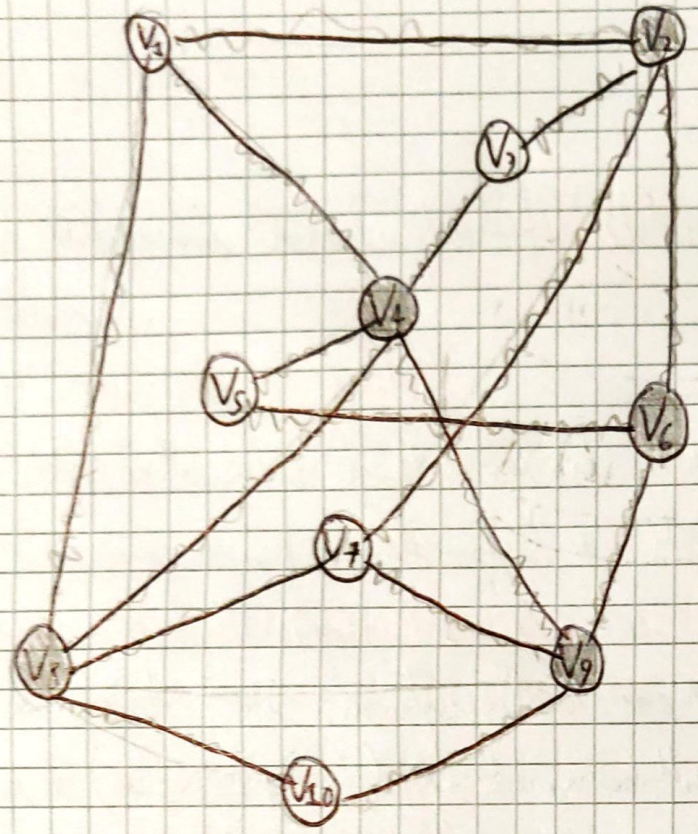


IS = 1, 3, 5 / VC = 2, 4, 6

ESERCIZIO 8: CAP 1

IS = 0

VC = 0



TROVARE IS E VC:

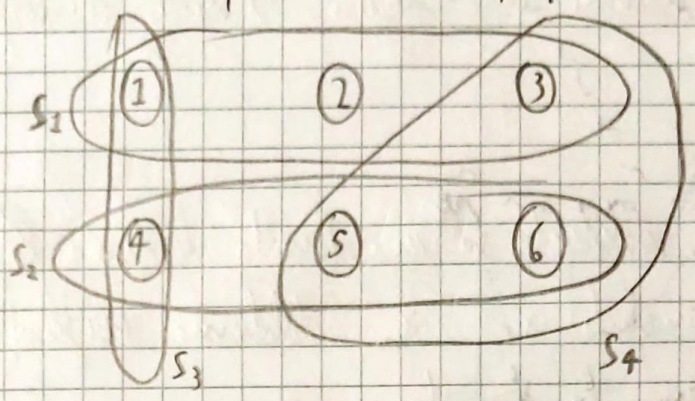
Un insieme indipendente $S = \{v_1, v_3, v_5, v_7, v_{10}\}$, mentre un vertex cover $S/V = \{v_2, v_4, v_6, v_8, v_9\}$.

ESERCIZIO 9: CAP 1

TRONA (PAG: 20 - 21)

SETCOVER = SET PACKING ?

Il problema SC non è equivalente a SP, possiamo dimostrare ciò con un contro esempio:



Gli insiemi S_1 e S_2 sono una soluzione per il problema SET PACKING, mentre i restanti insiemi, ovvero S_3 e S_4 non sono soluzione per SET COVER perché essi non coprono tutti gli elementi di U, ovvero l'elemento 2 rimane scoperto.

ESERCIZIO 5: (AP 1

TEORIA (PAG: 22/3) e TEORIA (PAG: 37/8)

$SAT \leq_T 3SAT$

~~a. RIDUZIONE POLINOMIALE~~

~~Per provare la polinomialità bisogna risolvere SAT sfruttando l'oracolo che risolve 3SAT, usando tempo polinomiale.~~

~~1. CONSIDERARE ISTANZA SAT (1° PROBL):~~

~~Consideriamo un'istanza di SAT, costituita da una formula booleana ϕ espressa come AND di OR su un insieme di variabili $x_1, \dots x_n$, dobbiamo assegnare delle variabili dei valori in modo da rendere vera la formula.~~

Sappiamo che CIRCUIT SAT è NP-COMPLETO (TEOREMA 1.62, PAG: 38) e che SAT è NP (è facile dare un verificatore polinomiale). Sappiamo anche che $CIRCUITSAT \leq_T 3SAT$ (PAG: 26), quindi anche 3SAT è NP-COMPLETO.

Possiamo sfruttare la proprietà di NP-COMPLETO, ovvero un problema X è NP-COMPLETO se:

1) $X \in NP$

2) Ogni problema $Y \in NP$ si riduce a X, cioè $Y \leq_T X$

Quindi 3SAT è NP-COMPLETO, $SAT \in NP$ e quindi $SAT \leq_T 3SAT$

ALTERNATIVAMENTE CON LA TRANSITIVITÀ $SAT \leq_T CIRC.SAT \leq_T 3SAT$

ESERCIZIO 10: CAP 1

TEORIA (PAG: 23)

4SAT $\leq_T$ INDIPENDENT SET

0. DESCRIZIONE RIDUZIONE:

Per provare la riduzione bisogna risolvere 4SAT sfruttando l'oracolo che risolve IS in tempo polinomiale.

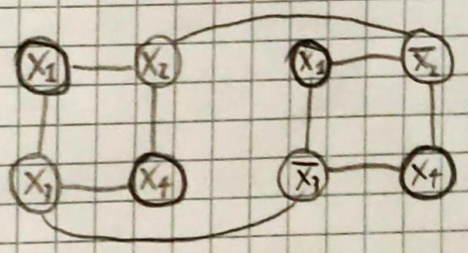
1. CONSIDERARE ISTANZA 4SAT (1 PROBL.)

Consideriamo un'istanza di 4SAT, costituita da una formula booleana $\emptyset = C_1 \cdots C_k$ dove ogni clausola C_i è costituita da 4 letterali delle variabili $x_1, \dots x_n$.
Bisogna trovare un assegnamento alle variabili in modo da rendere vera la formula $\emptyset$.
~~provo a assegnare ad almeno un letterale per clausola il valore vero.~~

2. COSTRUIRE ISTANZA IS (2° PROBL.) IN RELAZIONE A 4SAT (1° PROBL.) PER USARE L'ORACOLO:

Per usare l'oracolo bisogna costruire un'istanza di IS equivalente a 4SAT.
L'istanza di IS è costituita da un grafo $G=(V,E)$ contenente $4k$ nodi (4 per ogni k clausole) ed ogni nodo corrisponde ad un letterale. I 4 vertici corrispondenti alla stessa clausola saranno uniti da 4 archi formando un quadrato, avremo k di questi quadrati, uno per ogni clausola. Aggiungeremo degli archi extra tra un letterale ed il suo complemento siccome non possono essere entrambi veri.

$$\emptyset = (\underset{1}{x_1} \vee \underset{1}{x_2} \vee \underset{1}{x_3} \vee \underset{1}{x_4}) \wedge (\underset{1}{x_1} \vee \underset{0}{\bar{x}_2} \vee \underset{0}{\bar{x}_3} \vee \underset{1}{x_4})$$



3. DIM SSE E RISOLVO ORACOLO:

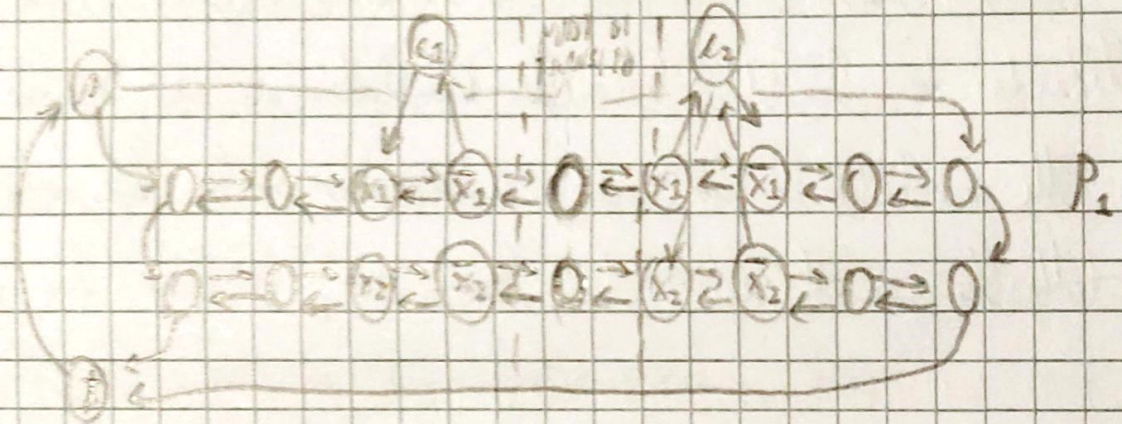
Il grafo così costruito ha questa proprietà: esiste un IS di taglia almeno k se e solo se la formula è soddisfacibile. I nodi del grafo corrispondenti ai letterali veri, formano un insieme indipendente. Mentre, se esiste un insieme indipendente di k vertici è possibile soddisfare la formula assegnando vero ai letterali corrispondenti a quei k vertici.
Usando l'oracolo per risolvere IS, se esiste un insieme indipendente di taglia k, allora la formula è soddisfacibile.

ESERCIZIO 14: CAP 1

TEORIA (PAG 29): HAMCYCLE

POSIZIONE DEI NODI "DI TRANSITO"

Quando si percorre un ciclo ham. bisogna passare per tutti i nodi del percorso P_i.
I nodi "di transito" servono per percorrere tutto P_i, siccome da c_i senza essi è impossibile andare su un altro percorso P_j.
Se, invece, ci sono nodi di transito ma non vengono percorsi, non ci sarà nessun ciclo.



ESERCIZIO 1): CAP 1

TEORIA: (PAG 25)

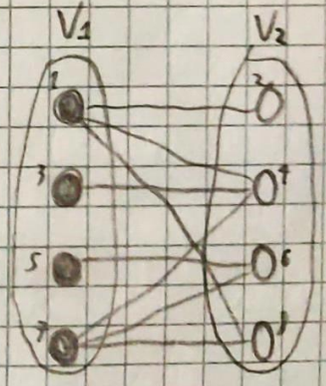
Una 2-colorazione del grafo G è definita come un funzione $f: V \to \{1, 2\}$ tale che per ogni arco (u, v) si ha che $f(v) \neq f(u)$.

Se esiste una 2-colorazione per G allora G è 2-colorabile.

Un grafo G è 2-colorabile se e solo se G è bipartito:

($\Rightarrow$) Se esiste una 2-colorazione del grafo G significa che esiste un arco (u, v), i cui vertici avranno $f(v) \neq f(u)$, cioè hanno colori diversi, pertanto possono essere raggruppati in 2 insiemi differenti (insieme di colore 1 e insieme di colore 2). Viceversa, se due vertici hanno lo stesso colore non esiste un arco tra loro, pertanto nei due insiemi non esistono archi tra i nodi appartenenti. Questo soddisfa la proprietà $V = V_1 \cup V_2$ ed ogni arco ha un vertice in V_1 e l'altro in V_2, pertanto G è bipartito.

($\Leftarrow$) Se esiste un grafo G bipartito, ovvero è possibile raggruppare i vertici in due insiemi differenti tale che $V = V_1 \cup V_2$ e in ogni insieme non esistono archi tra vertici, possiamo colorare tutti i vertici di V_1 di un colore e tutti i vertici di V_2 di un altro colore. Le proprietà di 2-colorabile sono soddisfatte siccome però qualsiasi arco (u, v) di E abbiamo che $f(u) \neq f(v)$, pertanto G è 2-colorabile.



ALGORITMO:

assegna v "esplorato" e $level(v) = 0$

crea una queue per eseguire una ricerca in profondità (BFS)

inserisci in queue un vertice sorgente v (root dell'albero)

WHILE finchè che la queue non è vuota DO:

rimuovi dalla queue il nodo anteriore e consideralo come v

FOR per ogni arco (v, u) DO

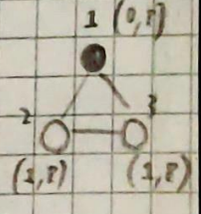
IF se u non è esplorato THEN assegna u come "esplorato"

assegna $level(u) = level(v) + 1$

inserire u nella queue

ELSE IF $level(u) = level(v)$ THEN G non bipartito

RETURN G è bipartito



PAG: 75 ESERCIZIO 1: (CAP 2)

TRONE (PAG 53)

FATTORIZZAZIONE DI N, QUANTI BIT BASTA ALLUNGARE N PER PERDERE IL VANTAGGIO (DIVISIONE FINO A N/2)?

Nel problema della fattorizzazione, tentando di dividere N per tutti i numeri più piccoli fino ad N/2 si dimezza l'insieme dei potenziali divisori di N.
Per perdere questo vantaggio, basta raddoppiare il valore di N, pertanto, in termini di codifica in bit, basta allungare N di un solo bit siccome rengono raddoppiati i possibili valori di N.

N = 2 bit

$$\left.\begin{matrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{matrix}\right\} 2^2 = 4$$

N = 3 bit

$$\left.\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{matrix}\right\} 2^3 = 8$$

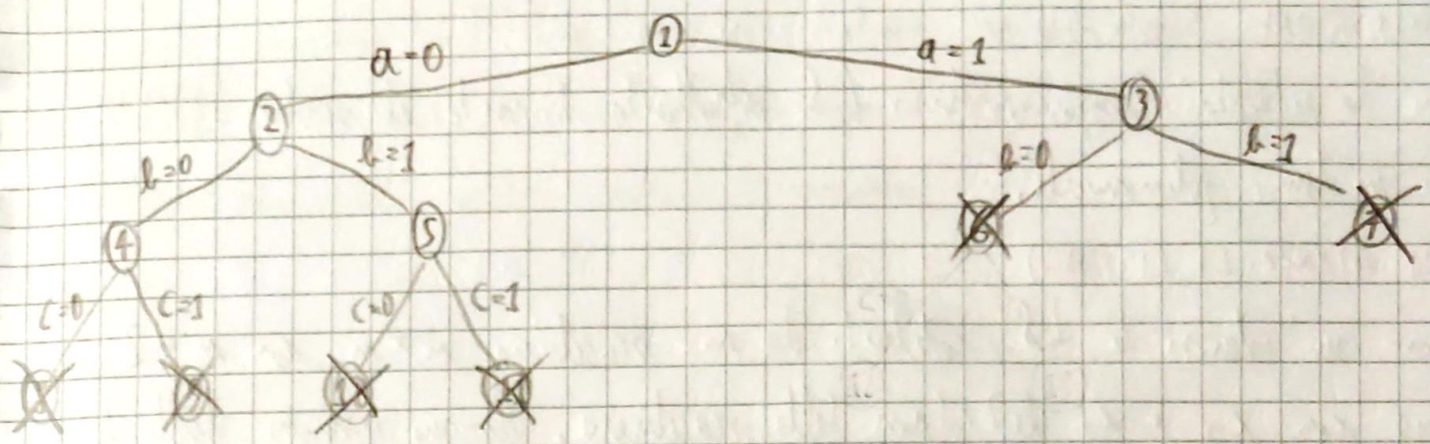
ESERCIZIO 1 PAG 75 (CAP. 2): L'esplorazione dello spazio di ricerca

Il vantaggio è che si dimezza, dato che lo spazio di ricerca viene ridotto della metà, perché anziché provare per tutti i numeri da 1 a n ci si ferma ad n/2.
Per determinati problemi si ha uno spazio di ricerca enorme e per problemi NP-hard l'unica soluzione è usare la ricerca esaustiva su tutto lo spazio, tutto ciò richiede troppo tempo.
Possiamo però usare degli accorgimenti che permettono di migliorare il tempo di esecuzione, come quello di fermarsi ad n/2. Questo però dipende dal problema.
Per perdere questo vantaggio (controllare solo n/2 numeri) basta allungare l'input di 1 bit perché se si aumenta la rappresentazione di un solo bit si riesce a raddoppiare in binario

ESERCIZIO 2: CAP 2

TRONI (MG 54)

ALBERO BACKTRACKING $\emptyset = (\neg a + b + \neg c + d) \cdot (\neg a + \neg b) \cdot (\neg a + b) \cdot (a + \neg c) \cdot (a + c)$



6. $\emptyset(1,0,c,d) = 0$ X

7. $\emptyset(1,1,c,d) = 0$ X

8. $\emptyset(0,0,0,d) = 0$ X

9. $\emptyset(0,0,1,d) = 0$ X

10. $\emptyset(0,1,0,d) = 0$ X

11. $\emptyset(0,1,1,d) = 0$ X

La formula $\emptyset$ non è soddisfacibile siccome non esiste assegnamento che rende vera la formula.

ESERCIZIO 3 (AP 2

TRONA (PAG: 57)

	2

LATINSQUARE $\leq_T$ SUDOKU DALL'ISTANZA →

0. DESCRIZIONE INDUZIONE

Per provare la riduzione bisogna risolvere LS sfruttando l'oracolo che risolve SUDOKU, tutto ciò in tempo polinomiale.

1. CONSIDERARE ISTANZA LS (1° PROBL.)

Consideriamo un'istanza di LS, costituita da una scacchiera $n \times n$, con $n = 2$, e indichiamo con c_1 e c_2 le 2 colonne della scacchiera. Bisogna riempire tutte le caselle con un numero compreso tra 1 e n ($n=2$) in modo che in ogni riga e colonna non ci siano numeri ripetuti.

2. COSTRUIRE ISTANZA DI SUDOKU (2° PROBL.) IN RELAZIONE A LS (1° PROBL.) PER USARE L'ORACOLO

Per usare l'oracolo di SUDOKU in modo da risolvere sia SUDOKU che LS, l'istanza di SUDOKU è costituita da una scacchiera $n^2 \times n^2$, con $n = 2$, ed ogni casella deve contenere un numero compreso tra 1 e n^2 ($n=2$), in modo tale che in ogni riga, colonna e sottoscacchiera non ci siano numeri ripetuti. L'istanza è la seguente: La trasformazione descritta avviene in tempo polinomiale.

	2
c_1	c_2

	3	4	2
	4	3	
3	1	2	4
4	2	1	3

3. ANALISI RISP. ORACOLO

L'istanza di SUDOKU così costruita, si ha che nelle celle vuote delle colonne c_i è possibile inserire solo i numeri tra 1 e n. Pertanto, chiamando l'oracolo di SUDOKU per risolvere l'istanza, le colonne c_i della soluzione di SUDOKU sono una soluzione che completa anche l'istanza del LATINSQUARE.

ESERCIZIO 4 CAP 2

TRONO (PAG: 56)

RESTRIZIONE LATIN SQUARE E ALGORITMO EFFICIENTE

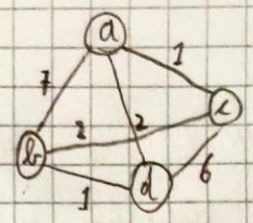
$n = 4$

1	2	3	4

Una possibile restrizione sull'input è quella di inserire una riga con numeri da 1 ad n. Considerando questa restrizione, un possibile algoritmo efficiente è effettuare uno shift ciclico sulle righe successive.

ESERCIZIO 5 CAP 2

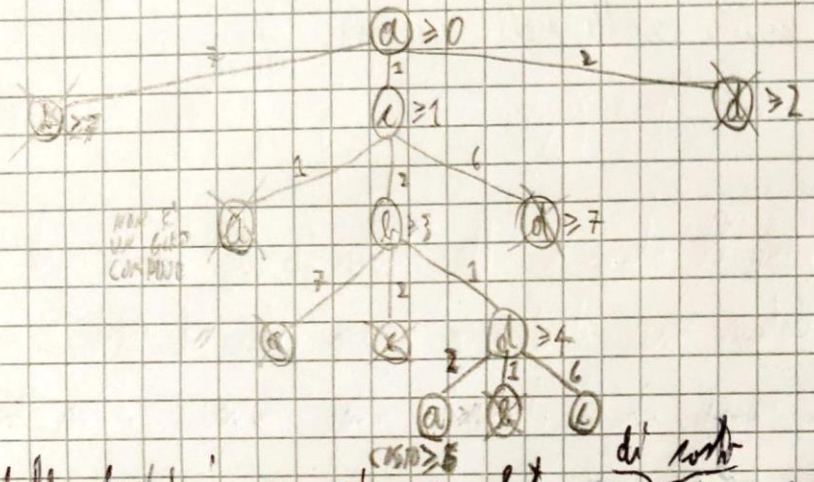
TEORIA (PAG: 60)



BRANCH AND BOUND PER TSP DEL GRAFO →

L'albero delle decisioni rappresenta tutti i possibili giri del grafo.

Tale albero ha un numero esponenziale di nodi in n dove n è il numero di nodi.



Seguendo l'albero delle decisioni, un giro completo di costo minimo è il giro a-c-b-d-a che ha costo 6.

ESERCIZIO 6 — CAP 2

TEORIA (PAG 68)

ALGORITMO BRANCH AND BOUND PER IND. SET E DISEGNARE L'ALBERO:

ALGORITMO: GREEDYIS (F):

$S = \{\}$; IS inizialmente vuoto;

WHILE F ha almeno un arco DO ← GREEDY

  Sia $e = (u, v)$ un arco di F tale che v è una foglia

  $S = S \cup \{v\}$

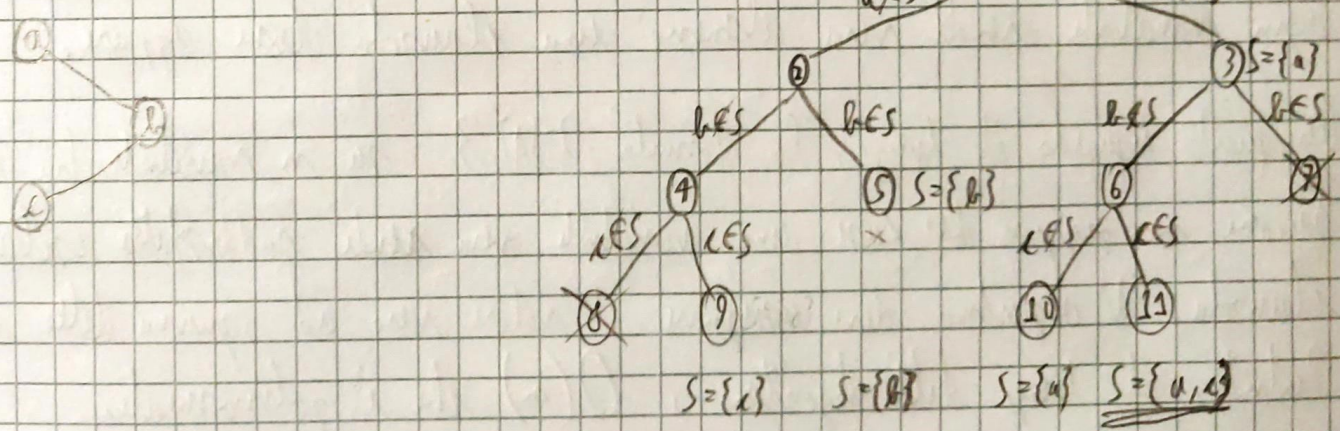
  Cancella da F i nodi u, v e tutti gli archi ad essi incidenti

RETURN S;

ALGORITMO BRANCH AND BOUND IS:

Ogni livello dell'albero rappresenta un vertice del grafo e ogni nodo ha 2 figli che rappresentano rispettivamente la non appartenenza e l'appartenenza all'insieme indipendente. Un determinato cammino rappresenta l'appartenenza o non appartenenza dei vertici, dalla root fino ad un nodo v, alla soluzione.

Esempio:



ESERCIZIO 7: CAP 2

TRONA (PAG 53)

3SAT-3 FORNIRE UN ALGORITMO POLINOMIALE?

Supponendo di avere una formula 3-SAT 3 come segue:

$$\emptyset(x_1, x_2, x_3) = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

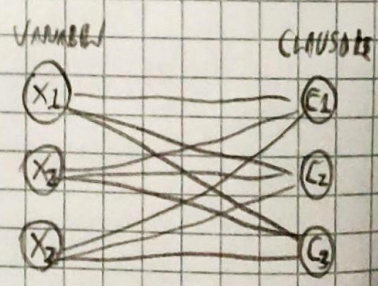
oppure

$$\emptyset(x_1, x_2, x_3, x_4) = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_3 \vee x_4 \vee x_1) \wedge (x_4 \vee x_1 \vee x_2)$$

L'algoritmo per trovare un assegnamento alla formula $\emptyset$ che la rende soddisfacibile opera come segue: assegna alla prima variabile un valore (0 o 1) che la rende vera. In questo modo almeno una clausola dove appare la prima variabile sarà soddisfatta. Successivamente, si rende vera la seconda variabile in una clausola non soddisfatta, in questo modo, almeno una clausola dove appare la seconda variabile è soddisfatta, e così via per le successive variabili.
In questo modo la formula $\emptyset$ 3-SAT 3 è sempre soddisfacibile, siccome ogni variabile rende vera almeno una clausola dove appare.

Per quanto riguarda il tempo, la formula 3-SAT 3, su n variabili ha n clausole. Siccome nei peggiori dei casi una variabile vera rende soddisfatta esattamente una clausola, l'algoritmo deve assegnare il valore vero ad ognuna delle n variabili. Pertanto, il tempo dell'algoritmo è $O(n)$ che è polinomiale.

OPPURE CON I GRAFI BIPARTITI:



Possiamo effettuare una matching del grafo per associare un valore alle variabili che rendono vera le clausole, ad esempio:

x_1 — C_1 => $x_1 = 1$

x_2 — C_2 => $x_2 = 1$

x_3 — C_3 => $x_3 = 0$

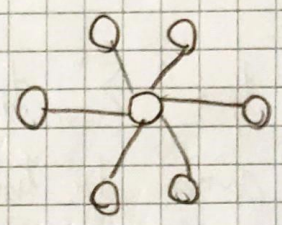
L'assegnazione rende vera $\emptyset$.
Non ci sono conflitti siccome una variabile rende vera almeno una clausola

PAG. 102 ESERCIZIO 1 (CAP 3)

TEORIA (PAG 79)

APPROXVC SOL. SUB-OTTIMALE:

Un grafo a stella è un grafo con un nodo collegato a tutti gli altri.
L'algoritmo APPROXVC restituisce sempre una soluzione sub-ottimale se gli viene dato un grafo a stella, siccome esso restituirà sempre un insieme ricoprente con 2 nodi, mentre quello ottimale restituirà sempre il nodo centrale.



$|OPT| = 1$

$|V| = 2$

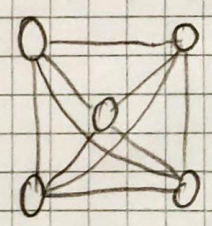
ESERCIZIO 2:

APPROXVC SOL OTTIMA:

L'algoritmo APPROXVC restituisce sempre una soluzione ottima se gli viene dato un grafo completo DISPARI, siccome esso restituisce un insieme ricoprente di $n-1$ nodi, ma anche la soluzione ottimale restituisce $n-1$ nodi.

$|OPT| = n-1$

$|V| = n-1$



ESERCIZIO 3: (AP)

TEORIA (C8/9)

ALGORITMO VC CON UN ALBERO:

ALGORITMO: GREEDY VS(F)

$S = \{\}$ è l'insieme ricoprente inizialmente vuoto

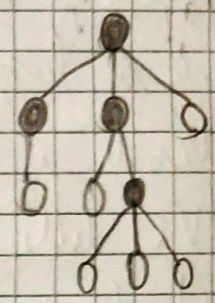
WHILE F ha almeno un arco DO

Sia $e = (u, v)$ un arco di F tale che v è una foglia ed u il padre

$S = S \cup \{u\}$ // inseriamo il padre in S

Cancellare da F i nodi u e v, e tutti gli archi ad essi incidenti

RETURN S



Sia S^* una soluzione ottima, abbiamo 2 casi:

1. $u \in S^*$, caso banale siccome u è il padre che copre i figli;
2. $u \notin S^*$, allora $v \in S^*$ siccome l'arco (u, v) è coperto.

Possiamo definire $S' = S^* \setminus \{v\} \cup \{u\}$, allora (u, v) rimane comunque coperto.

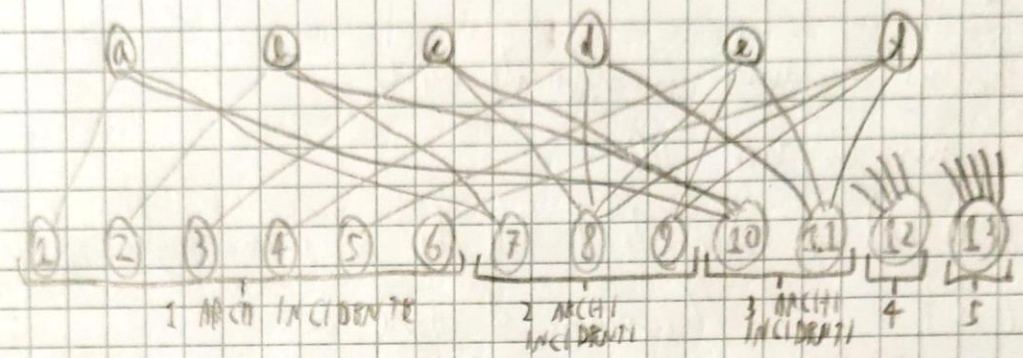
Creiamo S' perché v essendo foglia copre solo l'arco (u, v).

Per quanto riguarda l'efficienza, le visite e le cancellazioni di archi e nodi avvengono in modo non esponenziale, ovvero $|E| \leq n^2 \Rightarrow O(n^2)$.

ESERCIZIO 4: CAP 3

TRONE (PAG 79)

FORNIRE ESEMPIO PER VC ~~[illegible]~~ DOVE NON È 2-APPROSSIMATO



Scegliendo il nodo con il più alto numero di archi incidenti, VC sarà composto dai nodi in basso siccome hanno il maggior numero di archi incidenti, ma la soluzione ottimale è il VC composto dai nodi in alto, |VC| della soluzione è 13 ma |VC| della soluzione ottima è 6, pertanto non è 2-approssimato siccome è maggiore del doppio.

ESERCIZIO 5: CAP 3

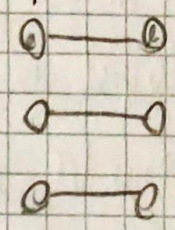
TROVA (PAG 79)

APPROX VC PER TROVARE IS, ANALIZZARE ALGORITMO:

ALGORITMO PER IS: usare APPROX VC → restituisce S

restituisce V \ S

Il fattore di approssimazione non è 2, perché $\overbrace{\text{APPROX VC è 2-approssimato}}^{\text{FATTO}}$, quindi avendo un grafo bipartito come segue:



APPROX VC restituisce tutti i nodi del grafo e pertanto $IS = \{\}$.

Dato il controesempio, questo approccio non è 2-approssimato, siccome con un grafo del genere non ha lo stesso fattore di approssimazione

Se consideriamo delle condizioni sul grafo, possiamo farlo diventare 2-approssimato:

Sia S^* una soluzione ottima per VC e $k = |S^*|$.

Sia S una soluzione di APPROX VC, implica che $k \leq |S| \leq 2k$.

Sia S' la soluzione di IS, ovvero $S' = V \setminus S$, implica che $n-k \leq |S'| \leq n-2k$

La cardinalità dell'IS ottimo è $n-k$. Quindi abbiamo che:

$$n-2k \leq |S'| \leq n-k$$

Il fattore di approssimazione è $\rho = \dfrac{\text{VALORE SOL. OTTIMA}}{\text{// APPROX}}$

$$= \frac{n-k}{n-2k} \leq 2$$

$$n-k \leq 2(n-2k)$$

$$n-k \leq 2n-4k$$

$$4k-k \leq 2n-n$$

$$3k \leq n$$

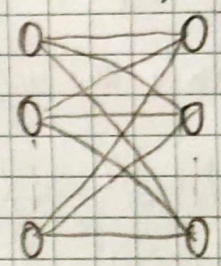
Quindi se il numero di nodi è molto più grande del valore della soluzione ottima (di un fattore 3) allora l'algoritmo per IS è 2-approssimato.

ESERCIZIO 6: CAP 3

TRONA (PAG 83)

L'ANALISI DI APPROX VC PUÒ ESSERE MIGLIORE?

Per stabilire se l'analisi di un algoritmo di approssimazione è la migliore possibile si possono cercare esempi di input per i quali l'algoritmo fornisce l'approssimazione dell'analisi. Nel nostro caso APPROX VC è 2-approssimato. Consideriamo un grafo bipartito completo con n pari (numero di nodi) in cui ognuno degli n/2 nodi di sinistra è collegato ad ognuno degli n/2 nodi di destra.



APPROX VC selezionerà tutti i nodi, mentre un ricoprimento ottimale selezionerà tutti gli n/2 nodi di destra o di sinistra. Pertanto, questo algoritmo, per questo grafo fornisce sempre una soluzione 2-approssimata.

Dunque, l'analisi fatta è stretta, cioè è la migliore possibile da quanto si sono casi in cui l'algoritmo fornisce soluzioni che hanno un fattore di approssimazione uguale a quello dell'analisi.

ESEMPIO 7: (AP 3)

TROVA (PAG: 86)

ALGORITMO WEIGHTEDVC CON $x_i \geq 0,75$, $x_i \geq 0,25$:

Il problema INTEGER PROGRAMMING è NP-COMPLETO, pertanto rinunciamo al vincolo $x_i \in \{0,1\}$ richiediamo, quindi, che $0 \leq x_i \leq 1$. Questo è un problema lineare risolvibile efficientemente.

~~Se definiamo~~ le copertura $S = \{i \mid x_i \geq 0,75\}$ o $S = \{i \mid x_i \geq 0,25\}$, ~~considerando un arco~~ $e = (i,j) \in E$, dalla soluzione del ~~problema di programmazione~~ lineare si ha che $x_i + x_j \geq 1$, ~~quindi è necessario~~ che o x_i o x_j siano maggiori o uguali a ~~0,75~~ (oppure 0,25 nel secondo caso), ~~almeno uno dei due sarà arrotondato a 1 e quindi almeno uno dei due vertici i e j sarà nell'insieme.~~

~~Pertanto, la differenza tra le due coperture siccome non appena x_i o x_j assumono un valore maggiore di 0, questi saranno arrotondati ad 1 e pertanto il nodo associato sarà selezionato.~~

~~Dal lemma 3.2.9 abbiamo che $w(S_{LP}) \leq 2 \cdot w(S^*_{IP})$.~~

~~Con $x_i \geq 0,75$ abbiamo che: $w(x^*_{IP}) \geq \frac{3}{4} w(x^*_{LP}) \Rightarrow w(S_{LP}) \leq \frac{4}{3} w(S^*_{IP})$~~

~~Con $x_i \geq 0,25$ abbiamo che: $w(x^*_{IP}) \geq \frac{1}{4} w(x^*_{LP}) \Rightarrow w(S_{LP}) \leq 4 w(S^*_{IP})$~~

~~Quindi per 0,75 l'approssimazione è migliore rispetto a 0,5 e 0,25.~~

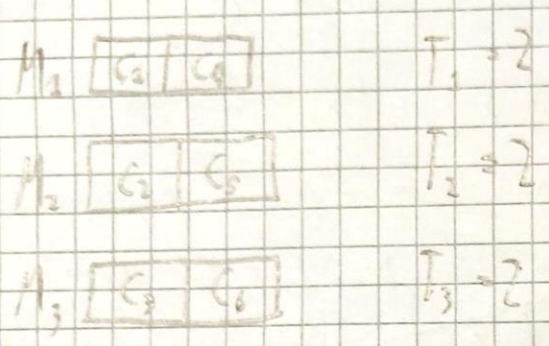
- Con $S = \{i \mid x_i \geq 0.25\}$ abbiamo che l'algoritmo sceglierà più nodi dell'ottimo (0.5) pertanto sarà sempre sottoottimale.
- Con $S = \{i \mid x_i \geq 0,75\}$ abbiamo che l'algoritmo sceglie meno nodi di 0.5 ma rischia di dare soluzioni sbagliate siccome potrebbe non considerare dei nodi importanti, in quel caso potrebbe non trovare una soluzione siccome la soglia è troppo alta.

ESERCIZIO 8: CAP 3

TROVA (PAG: 84)

SOLUZIONE OTTIMA PER APPROX LOAD IN ORDER:

Supponiamo di avere 3 macchine M_1, M_2, M_3 e un insieme di 6 compiti $C_1, \dots C_6$ e ognuno dei compiti necessita di tempo di esecuzione pari a $t_i = 1$:



COMPITI	TEMPO
C_1	1
C_2	1
C_3	1
C_4	1
C_5	1
C_6	1

Pertanto, per questo algoritmo, questa configurazione restituisce il valore ottimo, cioè la differenza tra il carico massimo e quello minimo è 0 proprio come l'ottimo.

In generale, quando il tempo necessario dei compiti è 1, l'algoritmo approssimato trova sempre l'ottimo.

ESERCIZIO 9:
TROVA (PAG: 87)

(AP 3)

DATO SOL. OTTIMA S^*, RIORDINARE I LAVORI DA DARE AD APPROXLOADINORDER PER RESTITUIRE L'OTTIMO:

Supponendo di avere la soluzione ottima per LOADINORDER, cioè il carico massimo ottimale T^*.
Avendo una serie di compiti $C_1, \dots C_n$ da assegnare alle macchine $M_1, \dots M_m$, possiamo riordinare i compiti, usando T^*, da dare ad APPROXLOADINORDER in modo che restituisca anch'esso la soluzione ottima.
Il riordinamento funziona come segue:

Avendo dei compiti $C_1, \dots C_n$ che necessitano di tempo t_j, ordiniamo questi in m blocchi e per ogni compito nei blocchi, la somma dei tempi è uguale a T^*.

$$\underbrace{C_1 \; C_2 \; C_3}_{t_1+t_2+t_3 \leq T^*} \; \underbrace{C_4 \; C_5}_{t_4+t_5 \leq T^*} \; \dots \; C_n$$

← sono m blocchi, dove m è il numero di macchine

In questo modo, all'esecuzione di APPROXLOADINORDER le macchine verranno caricate in modo tale che il carico massimo è quello ottimale.

ESERCIZIO 10: CAP 3

CONDURRE SPEDIZIONE (LOAD BALANCING)

(a) Consideriamo un insieme di pacchi con determinati pesi $w_1, \dots, w_n$ da caricare su dei furgoni con capacità C, ~~supponiamo di caricare i furgoni a metà $C/2$ e farli partire~~. Esempio:

$C = 10$

PACCHI	PESI
P_1	6
P_2	5
P_3	6
P_4	5
P_5	6
P_6	5
P_7	6
P_8	5

$F_1 = \{P_1\}$ $F_5 = \{P_5\}$

$F_2 = \{P_2\}$ $F_6 = \{P_6\}$

$F_3 = \{P_3\}$ $F_7 = \{P_7\}$

$F_4 = \{P_4\}$ $F_8 = \{P_8\}$

L'esempio, così descritto, non fornisce mai la soluzione ottima.

OPT = $F_1 = \{P_1\}$ $F_4 = \{P_6, P_8\}$

$F_2 = \{P_2, P_4\}$ $F_5 = \{P_5\}$

$F_3 = \{P_3\}$ $F_6 = \{P_7\}$

(b) ~~Consideriamo il caso in cui i furgoni vengono riempiti fino a $C/2$ e per tanto la soluzione sarà T, la soluzione ottima utilizzerà almeno la metà dei furgoni siccome li riempie del tutto, pertanto avremo che:~~

$$\cancel{\frac{T}{2} \le T^* \Rightarrow T \le 2T^*}$$

~~Quindi l'algoritmo è 2-approssimato.~~

L'algoritmo ottimo non può usare meno di $r = \frac{\sum_i^m a_i}{c}$ furgoni, anche se spendo tutti i furgoni sono pieni. Per ogni coppia di furgoni consecutivi, si ha che il carico totale dei due contenitori è superiore a c, in quanto si passa al secondo perché il carico del primo più l'oggetto considerato fa diventare il peso totale maggiore di c, pertanto non saranno più di $2r$ contenitori.

$\underbrace{F_1 \quad F_2}_{W \ge C}$ $\underbrace{F_3 \quad F_4}_{W' \ge C}$