**DOMANDE TEORIA**

1. **Si spieghi qual è il significato del coefficiente di costo ridotto di un arco fuori base nell’algoritmo del simplesso per il problema di flusso a costo minimo e si spieghi in quali parti dell’algoritmo si utilizzano i coefficienti di costo ridotto (limitarsi al caso con capacità illimitate). \*\* (1° APPELLO)**

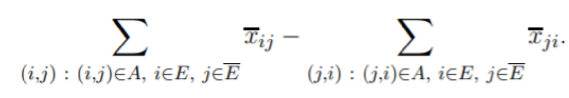
I coefficienti a costo ridotto sono associati ad ogni arco fuori dalla base ammissibile. Misurano quanto varia il valore obiettivo se incremento il valore del flusso su quell’arco fuori base a cui è associato il coefficiente, vengono utilizzati per la verifica della condizione d’ottimalità: se tutti i coefficienti degli archi fuori base sono non negativi allora la soluzione è ottima, e per la condizione d’illimitatezza: se l’aggiunta alla base di un arco fuori base con coefficiente di costo ridotto negativo si crea un ciclo allora il problema di costo minimo ha obiettivo illimitato ed infine si utilizza per scegliere quale arco far entrare in base.

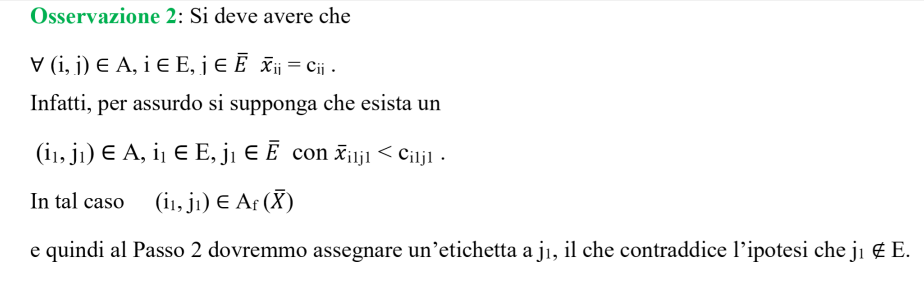
1. **Si descriva il significato dei coefficienti di costo ridotto, si descriva la procedura per il loro calcolo e si spieghi dove vengono utilizzati nell’algoritmo del simplesso per problemi di flusso a costo minimo con limiti di capacità sugli archi. \***

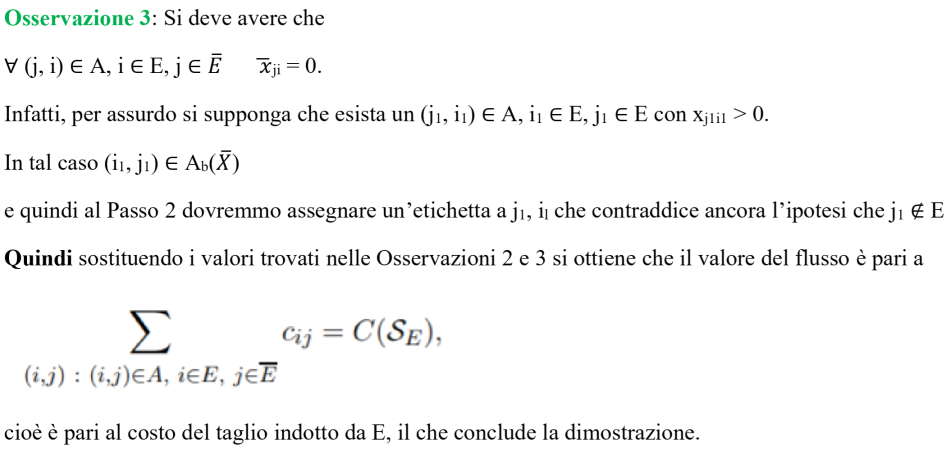
PRIMA PARTE DELLA DOMANDA PRIMA PARTE RISPOSTA 1. La procedura di calcolo dei coefficienti di costo ridotto per un problema di flusso a costo minimo con limiti di capacità sugli archi è identica al problema con capacità illimitata: si prende un arco fuori base e lo si aggiunge all’albero di supporto corrispondente alla base attuale. Si considera solo l’unico ciclo che si forma con tale aggiunta e si stabilisce il verso del ciclo quello dell’arco fuori base. Calcolare il coefficiente di costo ridotto sommando tra loro tutti i costi relativi agli archi attraversati dal ciclo nel loro stesso verso e sottraendo al risultato i costi degli archi attraversati dal ciclo in senso opposto. La presenza di un limite di capacità sugli archi implica l’esistenza di un nuovo insieme N1 che rappresenta gli archi fuori base con flusso pari alla capacità, inoltre per soddisfare la condizione di ottimalità i coefficienti di costo ridotto degli archi in N1 devono essere non positivi (se le disuguaglianze sono strette la soluzione ottima è unica), per quanto riguarda il criterio d’illimitatezza: esso viene rispettato se tutti gli archi hanno capacità limitata. Per la scelta dell’arco da far entrare in base bisogna scegliere il valore massimo tra il massimo coefficiente di costo ridotto degli archi fuori base di N1 e quelli cambiati di segno in N0, per quanto riguarda l’arco uscente dalla base se l’arco entrante appartiene ad N1 si riduce il suo valore da dij a dij − ∆ mentre sugli archi del ciclo che si forma con l’aggiunta di tale arco i flussi sono aggiornati in questo modo: il flusso viene diminuito di ∆ lungo gli archi attraversati secondo il proprio verso dal ciclo ed incrementato di ∆ lungo gli archi attraversati dal ciclo in verso opposto e l’arco passa da N1 ad N0.

1. **Si dimostri la correttezza dell’algoritmo di Ford-Fulkerson per il problema di flusso massimo e di taglio a costo minimo. \*\*\*\*\*\*\***

Al momento della terminazione dell’algoritmo si ha S ∈ E (il nodo S viene sempre etichettato al Passo 1) e D ∉ E (altrimenti dovremmo andare al Passo 3 ed aggiornare il flusso attuale). Quindi l’insieme E induce effettivamente un taglio. Se riusciamo a dimostrare che esso coincide con il valore del flusso uscente da S, avendo già osservato che il costo di ogni taglio è non inferiore al valore del flusso massimo, possiamo concludere che esso è il taglio a costo minimo e il flusso attualmente uscente da S è il massimo possibile. Il flusso uscente da S coincide con tutto il flusso che dai nodi in E viene spostato verso i nodi nel completamento di Ẽ = V \ E meno il flusso che va in senso opposto, quindi pari a:







1. **Si dimostri la correttezza dell’algoritmo greedy per il calcolo dell’albero di supporto a peso minimo. \*\*\*\*\*\***

Supponiamo per assurdo che esista un albero di supporto T′ = (V, ET′) a peso minimo con peso inferiore a T = (V, ET ), ovvero quello restituito dall’algoritmo greedy. Indichiamo con e\_h l’arco a peso più piccolo tra quelli in ET \ ET′.

Andiamo ad aggiungere e\_h a ET′. In tal caso si forma esattamente un ciclo che deve contenere almeno un arco e\_r e non a ET (gli archi in ET formano un albero di supporto e quindi non possono generare cicli).

Si deve avere che w\_e\_r ≥ w\_e\_h , altrimenti l’algoritmo greedy avrebbe selezionato e\_r al posto di e\_h (e\_r non forma cicli con gli archi in ET selezionati fino al momento in cui viene inserito e\_h se e\_h è il primo degli archi in ET che non fanno parte di ET′).

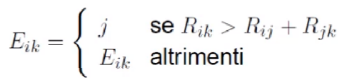
Ma allora possiamo togliere e\_r e sostituirlo con e\_h in modo da ottenere un albero di supporto a peso non superiore rispetto a T′. Iterando questo ragionamento, possiamo sostituire tutti gli archi in ET′ \ ET con archi in ET senza mai aumentare il peso di T′ fino a riottenere l’albero T con peso non superiore a T′, il che contraddice l’ipotesi iniziale.

1. **Si descrivano le classi di problemi P, NP e NP-completi e i legami esistenti tra queste classi, si dica che cosa potremmo concludere su di essi nel caso sapessimo che P≠NP. \* \* (2° APPELLO)**

Dato un problema di ottimizzazione R, questo appartiene alla classe P se e solo se esiste un algoritmo A di complessità polinomiale che lo risolve (ES. MST e SHORTEST PATH), appartiene alla classe NP se nota la soluzione ottima, il valore ottimo può essere calcolata in tempi polinomiali (ES. CLIQUE ed TSP). Tutti i problemi di P fanno parte di NP ovvero P ⊆ NP. Un problema R è NP-COMPLETO se R∈NP e per ogni problema Q∈NP, esiste una riduzione polinomiale di Q in R. (ES. CLIQUE, KNAPSACK, TSP) Nel caso P≠NP, allora non esisterebbero algoritmi di complessità polinomiale per tali problemi

1. **Si descriva l’algoritmo di Floyd-Warshall, si dica qual è la sua complessità, quale può essere il suo output e si spieghi come, eventualmente, ricostruire un cammino minimo tra una coppia di nodi al termine dell’esecuzione dell’algoritmo. \***

L’algoritmo di Floyd - Warshall permette la risoluzione dei problemi di tipo shortest path, trovare il cammino elementare di costo minimo orientato da un nodo s a un nodo t di un grafo orientato, l’algoritmo ha complessità O(n^3). Si inizializza una matrice R in cui sulla diagonale verranno messi valori pari a +infinito e nelle altre posizioni la matrice assume la distanza da i a j, viene inoltre inizializzata una matrice E di etichette in cui ogni elemento avrà come valore iniziale un trattino. Si esegue l’operazione di triangolazione sulla matrice R con j fissato (ovvero per ogni coppia di k appartenente ad un insieme di numeri interi da 1 a n sempre diversi da j viene confrontato l’attuale valore R\_ik con la somma dei valori R\_ij e R\_jk se R\_ik è <= della somma rimane invariato altrimenti viene aggiornato con il risultato della somma) e bisogna aggiornare la matrice E come segue



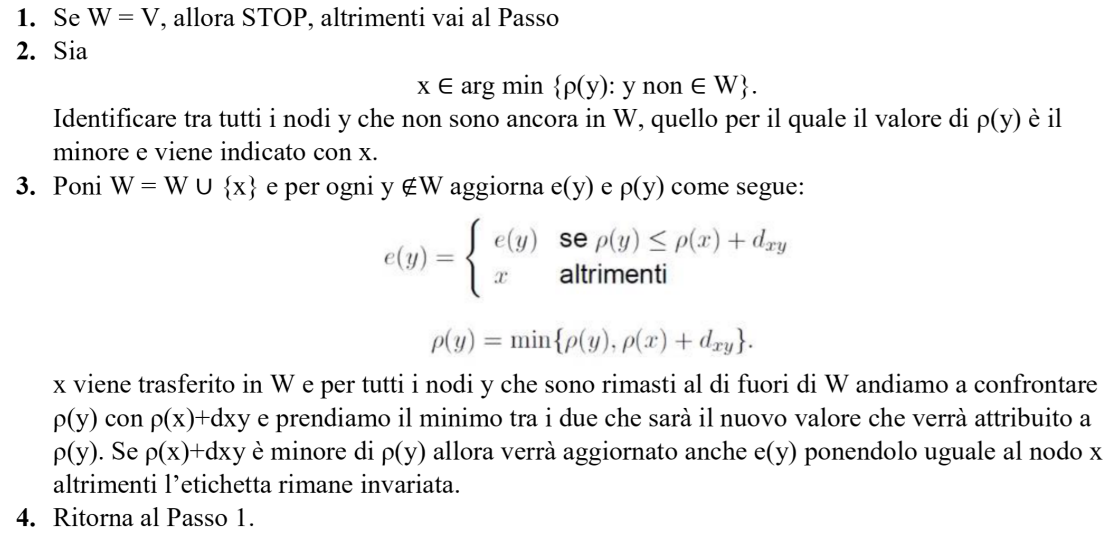
Se j=n o esiste R\_ii<0 allora STOP altrimenti si pone j+=1 e si ripete l’operazione di triangolazione come prima. Per ricostruire i cammini minimi bisogna utilizzare la matrice E, se vogliamo ricostruire un cammino tra x ed y vediamo la posizione (x,y) della matrice E: supponiamo di avere z -> x … z … y adesso bisogna determinare i valori tra x e z e z ed y come fatto prima fino a trovare un percorso completo.

1. **Si illustri l’algoritmo di Dijkstra per il cammino a costo minimo, si spieghi il significato degli insiemi e delle funzioni che appaiono in esso. Si dica sotto quali ipotesi è corretto e se ne dia la complessità. \***

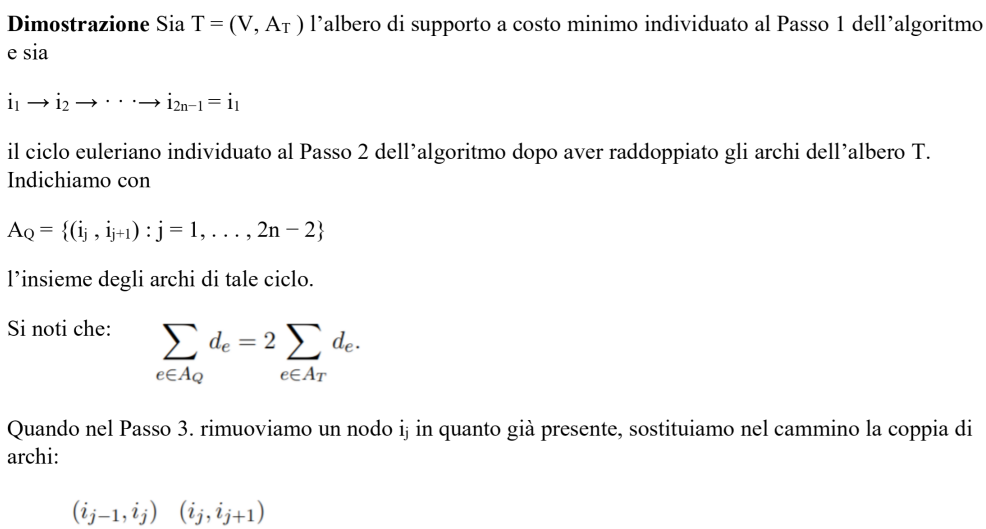
L’algoritmo di Dijkstra è valido solo se tutte le distanze del grafo sono positive, restituisce le distanze del grafo da un nodo s fissato, ha complessità pari a O(n) con n numero di nodi.

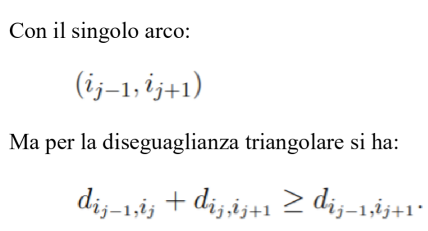
Inizializzazione: porre W = {s}, ρ(s) = 0, e(s) = - , ρ(y) = d\_sy, e(y) = s

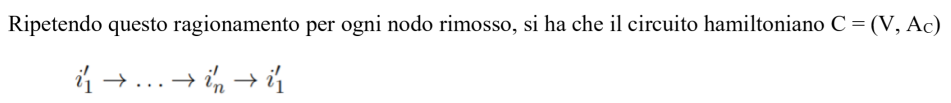
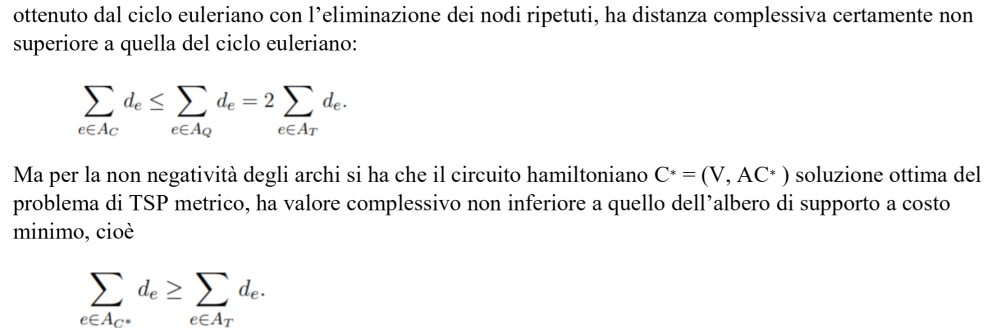
La funzione ρ associa un numero ad ogni nodo ad ogni iterazione: la lunghezza del cammino minimo passando da s ad y passando solamente attraverso i nodi che sono in W per le y che sono al di fuori di W, mentre per le y dentro W restituisce la lunghezza minima tra tutti i possibili cammini tra s ed y. La funzione e ha l’obiettivo di identificare nel cammino ( lungo ρ(y) ) qual è il nodo che precede y.

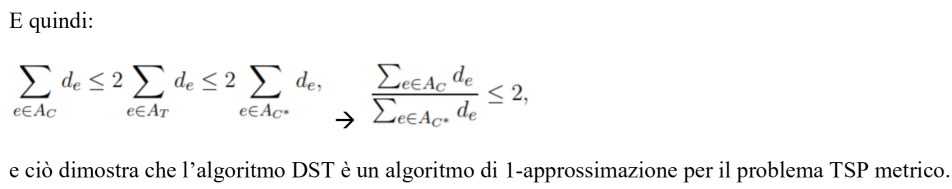


1. **Si dimostri che l’algoritmo Double Spanning Tree è un algoritmo di 1-approssimazione per il problema TSP metrico. \*\*\***

****

****

****

****

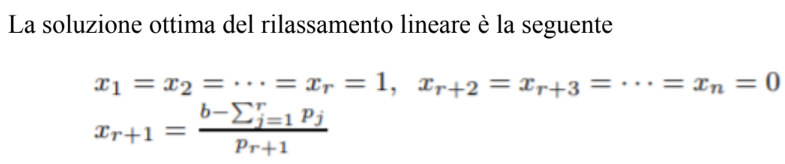
1. **Dopo aver spiegato che cos’è un upper bound relativo a un sottoinsieme della regione ammissibile di un problema di massimo, si descriva la procedura per il calcolo di un upper bound per il nodo radice nel problema KNAPSACK e si spieghi come si effettua il branching del nodo radice. \***

Dato un sott’insieme T della regione ammissibile l’upper bound è un valore U(T) con la seguente proprietà:

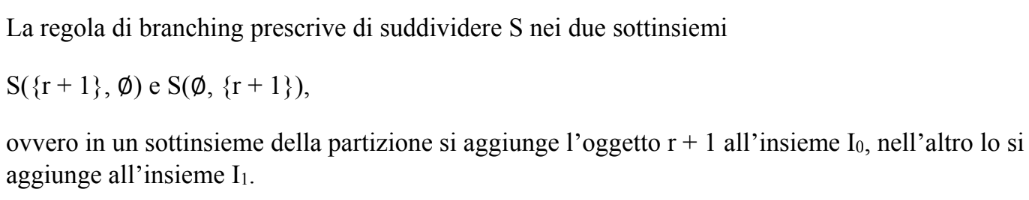
U(t)>= f(x) per ogni x∈T

Per calcolare l’upper bound di un nodo radice per un problema KNAPSACK occorre risolvere il rilassamento lineare sfruttando la funzione ausiliaria x\_i (1 se l’oggetto è inserito 0 altrimenti).

Occorre prima di tutto riordinare i valori dati dal problema in ordine decrescente per rapporto valore/peso, si calcolano i valori b-p1, b-p1-p2 etc. fino ad arrivare al primo valore negativo.



Se non si arriva ad un valore negativo nel punto precedente significa che la soluzione è inserire tutti gli oggetti nello zaino



1. **Si illustri il funzionamento dell’algoritmo MST-1 per il problema dell’albero di supporto a peso minimo, si spieghi il significato delle funzioni e degli insiemi in esso utilizzati e si dica qual è la sua complessità. \*\***

L’algortimo ha complessità O(|V|^2)

*inizializzazione:* Scegliere un nodo v\_1∈V, porre U = {v\_1} E\_T = 0

e c(v) = v\_1 con v∈V\{v\_1}, c associa ad ogni nodo diverso dal nodo v\_1 il nodo V\_1 stesso.

*condizione di STOP*: U=V

*passo 1:* selezioniamo tra tutti i nodi che sono in V\U quello in cui il peso dell’arco c(v) è minimo e lo indichiamo con v segnato

*passo 2:* Aggiungiamo ad U il nodo v segnato e a E\_T un nuovo arco che ha come estremi v segnato e c di v segnato

*passo 3:* Per tutti i nodi rimasti al di fuori di U, se v segnato è più vicino a v rispetto al nodo più vicino dell’iterazione precedente c(v) porre c(v) uguale a v segnato e ripetere l’algoritmo controllando nuovamente la condizione di stop.

1. **Dato un problema di ottimizzazione (o un equivalente problema di riconoscimento) si dia una definizione rigorosa di dimensione di un’istanza del problema e di algoritmo di risoluzione del problema con la relativa complessità. Si spieghi quando il problema appartiene alla classe P e quando appartiene alla classe NP. \***

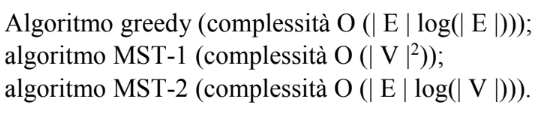
Dimensione istanza I → dim(I) = quantità di memoria necessaria per memorizzare (in codifica binaria) l’istanza I.

Procedura di risoluzione o algoritmo A per il problema.

numopA(I) = numero di operazioni elementari eseguite da A per risolvere I.

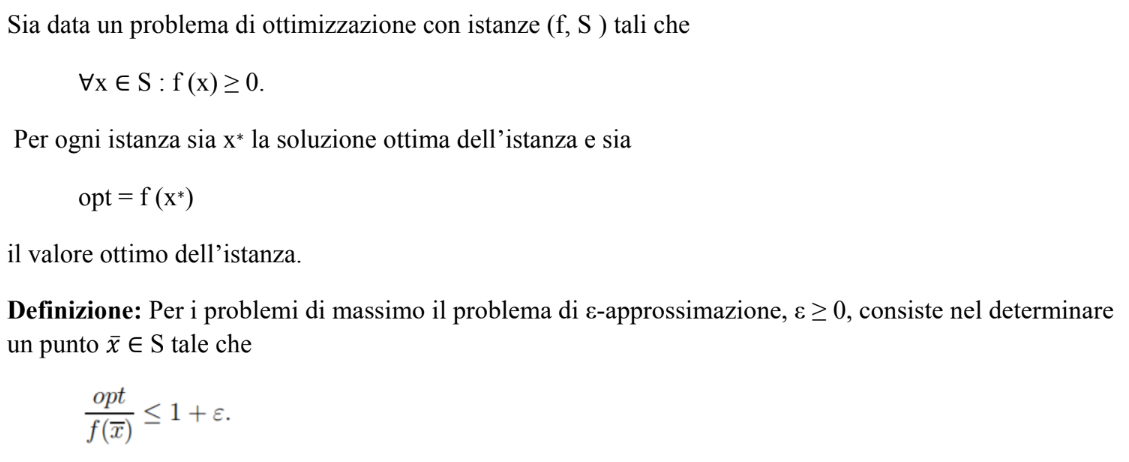
L’analisi worst case definisce il tempo t A ( k ) necessario all’algoritmo A per risolvere istanze di dimensione k come il massimo tra tutti i tempi di esecuzione di istanze di dimensione k.

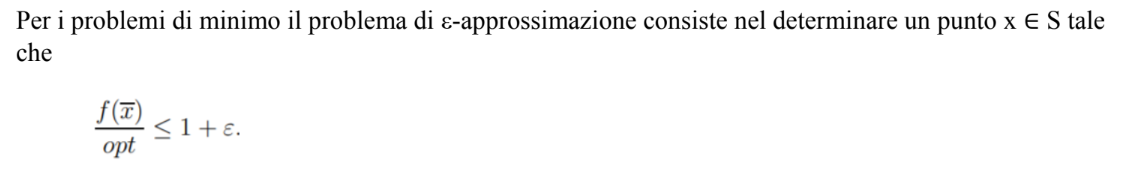
Si dice che la funzione tA(k) = O(g(k)), ovvero che tA(k) è dell’ordine di grandezza della funzione g(k), se esiste una costante u > 0 tale che tA(k) ≤ ug(k). L’algoritmo di Dijkstra richiede un numero di operazioni O (n^2) mentre Floyd-Warshall richiede un numero di operazioni O (n^3)

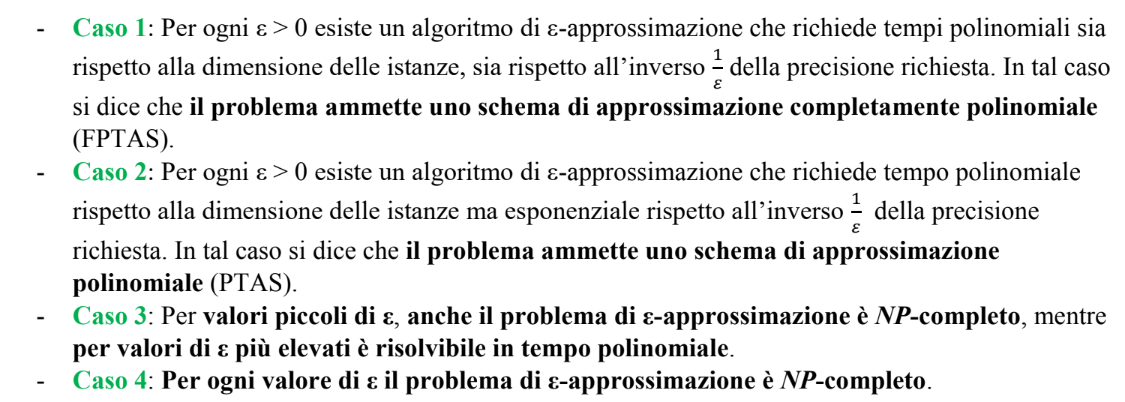


Dato un problema di ottimizzazione R, questo appartiene alla classe P se e solo se esiste un algoritmo A di complessità polinomiale che lo risolve (ES. MST e SHORTEST PATH), appartiene alla classe NP se nota la soluzione ottima, il valore ottimo può essere calcolata in tempi polinomiali (ES. CLIQUE ed TSP).

1. **Dopo aver dato la definizione di problema di ε-approssimazione per un problema di ottimizzazione, si illustrino i quattro diversi gradi di difficoltà dei problemi di ottimizzazione NP-completi basati sulla difficoltà dei rispettivi problemi di approssimazione. \***







KNAPSACK fa parte del caso 1, mentre TSP del caso 4

1. **Si illustri la prima iterazione dell’algoritmo ungherese per il problema di assegnamento, descrivendo le operazioni eseguite sulla tabella iniziale dei costi (indicata con T0 a lezione), le operazioni sulla tabella risultante (indicata con T2 a lezione) e il modo in cui viene (eventualmente) generata una nuova tabella T3. \***

Troviamo i minimi delle colonne di T0, sottraggo i minimi a T0 per ottenere T1, troviamo i minimi delle righe di T1,sottraggo i minimi a T1 per ottenere T2.

Per ottenere T3 devo individuare un sott’insieme delta di cardinalità massima degli 0 della matrice T2 tale che presi due elementi qualsiasi di delta essi sono indipendenti. Per calcolare delta occorre risolvere un problema di matching a cardinalità massima, se delta <n (cardinalità massima) occorre aggiungere un’ulteriore trasformazione della matrice fino ad arrivare al delta = n. Per ottenere altri zeri con le ulteriori trasformazioni bisogna prendere il numero più piccolo della matrice precedente e sottrarlo agli elementi della matrice per i quali non passano linee, e aggiungere questo numero agli elementi in cui passano due linee.