Relazione Progetto Algoritmi

# Introduzione:

Il progetto tratta la gestione delle risorse umane di un’azienda, in particolare la gestione di alcuni progetti e degli impiegati che devono partecipare a questi.

In ingresso abbiamo

* L’elenco di tutte le competenze che possiedono gli impiegati o richiedono i progetti.
* Un elenco di progetti di cui vengono specificate la durata, la scadenza e il guadagno; ad ogni progetto viene anche associato l’elenco delle competenze necessarie per essere svolto, per ogni competenza viene specificato il livello minimo necessario.
* Un elenco di impiegati e per ognuno di questi l’elenco delle competenze possedute con il relativo livello.

A partire da questi dati vengono effettuate le seguenti analisi.

# Analisi delle competenze:

## Problema:

Vogliamo studiare per ciascuna competenza quanti impiegati possiedono un tale livello di competenza e quanti progetti richiedono un tale livello di competenza, cioè ad ogni tipo di competenza devo associare due elenchi

* Il primo che riporta per ogni livello quanti impiegati possiedono il tale livello di competenza
* Il secondo che riporta per ogni livello quanti progetti richiedono il tale livello di competenza

## Struttura Dati:

Quindi dobbiamo creare una struttura dati che contenga le informazioni

* Nome della competenza
* Elenco di livelli degli impiegati
* Elenco di livelli dei progetti

Ogni elemento di questa tipologia deve poi essere salvato e ordinato alfabeticamente rispetto al Nome della competenza.

Quindi viene naturale pensare ad un albero di ricerca binaria, dal momento che può essere velocemente ordinato alfabeticamente.

### nodo:

Il nome della competenza è stato banalmente salvato in una stringa.

I due elenchi di livelli vengono invece salvati ognuno in una tabella, infatti una volta inseriti questa non viene mai modificata e la lettura di uno specifico elemento di una tabella (conoscendone l’indice) si può ottenere in tempo costante.

## Analisi spaziale:

Siano

* = numero dei livelli per una data competenza
* nc = numero di competenze totali

Per ogni competenza abbiamo due tabelle di nc interi, che quindi occupano , il nome della competenza è una stringa e quindi di valore costante. Quindi occupiamo .

## Analisi temporale:

L’unica operazione che viene svolta, a parte il salvataggio in memoria, è la stampa dell’albero binario.

Siano

* m = numero di livelli in un nodo
* n = numero di nodi, ovvero il numero di competenze

L’algoritmo di stampa prevede che per ogni nodo, cioè ogni competenza, vengano stampati tutti i livelli, quindi avremo un costo temporale di , infatti abbiamo

# Calendario:

## Problema:

Vogliamo costruire un Calendario che indichi giorno per giorno, se nel giorno corrente inizia o termina un progetto. Quando ci sono più progetti che iniziano o terminano nello stesso giorno vogliamo che vengano visualizzati in ordine alfabetico.

## Strutture Dati:

Per creare un Calendario che mostri solo i giorni in cui effettivamente è attivo un progetto dobbiamo salvare

* il giorno in cui inizia il primo progetto
* il giorno in cui termina il primo progetto

Infine dobbiamo salvare i progetti in una struttura dati adeguata; dal momento che i progetti devono essere visualizzati in ordine alfabetico ha senso salvarli in un albero di ricerca binaria strutturato nel seguente modo

### Nodo:

Il nodo dell’albero ha i seguenti argomenti

* una stringa in cui salvare il nome del progetto
* un interno in cui salvare l’inizio del progetto
* un intero in cui salvare la scadenza del progetto

## Analisi spaziale:

Sia

* np = numero di progetti

Abbiamo salvato np nodi formati da tre campi che occupano spazio costante, in più anche altri due interi per salvare il giorno di inizio e termine del Calendario, anche questi occupano spazio costante, perciò lo spazio occupato sarà

## Analisi Temporale:

Definiamo le quantità che useremo, siano

* ng = numero di giorni
* np = numero di progetti

Ora possiamo analizzare la Stampa del Calendario

1. Prima di tutto dobbiamo calcolare quanto vale ng
2. Dobbiamo stampare la parola “Calendario”, che ha un costo temporale di
3. Per ogni giorno dobbiamo fare
   1. Stampo il giorno corrente, costo
   2. Stampo quanti progetti sono attivi nel giorno corrente, che ha un costo , perché devo scorrere tutti i nodi dell’ABR contenente i progetti per controllare se il progetto è attivo nel giorno corrente, il controllo ha un costo di perché è un semplice confronto fra due interi.
   3. Stampo i nomi dei progetti che iniziano nel giorno corrente, che ha un costo , perché devo scorrere tutti i nodi dell’ABR contenente i progetti per controllare se il progetto inizia nel giorno corrente, il controllo ha un costo di perché è un semplice confronto fra due interi.
   4. Stampo i nomi dei progetti che terminano nel giorno corrente, che ha un costo , perché devo scorrere tutti i nodi dell’ABR contenente i progetti per controllare se il progetto termina nel giorno corrente, il controllo ha un costo di perché è un semplice confronto fra due interi.

Quindi il costo temporale è

# Relazione impiegato-progetti:

## Problema:

Vogliamo creare e visualizzare una relazione tra impiegato e progetti; un impiegato è in relazione con dei progetti se vi può partecipare, cioè se possiede almeno una delle competenze richieste dal progetto, con livello superiore o uguale al minimo richiesto.

Gli impiegati e i progetti devono essere visualizzati in ordine alfabetico, con ordine prima gli impiegati e poi i progetti a cui può partecipare un impiegato.

## Strutture Dati:

Dal momento che i dati devono essere visualizzati in ordine alfabetico, si è scelto di creare due alberi di ricerca binaria annidati. Il primo è fatto nel seguente modo

### Nodo impiegato:

Il nodo ha i seguenti argomenti

* Il nome dell’impiegato viene salvato in una stringa di caratteri
* L’elenco di progetti a cui può partecipare l’impiegato è salvato in un albero di ricerca binaria

### Nodo progetti:

Il nodo ha un unico argomento

* Il nome del progetto viene salvato in una stringa di caratteri

## Analisi Spaziale:

### Definiamo

* = numero di progetti al variare dell’impiegato i
* ni = numero di impiegati

Per analizzare il costo spaziale partiamo dall’interno, ogni elenco di progetti è contenuto in un albero che è formato da nodi, ogni impiegato ha associato un elenco di progetti e gli impiegati vengono salvati in un albero con ni nodi, allora il costo temporale di questa struttura dati è che è approssimativamente uguale a .

## Analisi Temporale:

Definiamo le quantità che useremo, siano

* hi = altezza dell’ABR con i nomi degli impiegati
* hp = altezza dell’ABR con i nomi dei progetti
* nci = numero competenze impiegato
* ncp = numero competenze progetto
* ni = numero impiegati
* np = numero progetti
* npi = numero progetti per impiegato

Ora possiamo analizzare l’inserimento degli impiegati, e dei progetti a cui partecipano, nell’ABR

1. Per ogni impiegato (in tutto ci sono ni impiegati)
   1. Inseriamo il nome dell’impiegato nell’ABR
   2. Per ogni competenza dell’impiegato corrente (in tutto ci sono nci competenze)
      1. Salvo la competenza dell’impiegato corrente
      2. Scorro ogni competenza richiesta da ogni progetto (quindi ncpnp)
         1. Salvo la competenza del progetto
         2. Se c’è corrispondenza fra la competenza dell’impiegato e la competenza del progetto
            1. Se il livello della competenza dell’impiegato è maggiore rispetto al livello della competenza del progetto

Inserisco il progetto dentro all’elenco di progetti a cui partecipa l’impiegato corrente

Quindi il costo temporale è

Ora calcoliamo il costo temporale della stampa

1. Per ogni impiegato dell’ABR
   1. Stampo il nome dell’impiegato
   2. Conto a quanti progetti può partecipare un impiegato
   3. Stampo il numero di progetti
   4. Per ogni progetto a cui può partecipare un impiegato (npi)
      1. Stampo il nome del progetto

Quindi il costo temporale è

# Relazione impiegato-impiegato:

## Problema:

Vogliamo creare e visualizzare la relazione fra impiegati, per cui due impiegati sono in relazione se possono partecipare ad uno stesso progetto. Vogliamo visualizzarli come coppie di impiegati in ordine alfabetico.

Quindi quello che vogliamo fare è prendere ciascun impiegato e cercare se fra gli altri impiegati c’è un progetto in comune. Quindi ci servono due strutture dati

1. La prima serve per accedere velocemente alla memoria che contiene i nomi degli impiegati e i nomi dei progetti, quindi deve essere una struttura che occupa poco spazio e per cui dato un indice è facile accedere alla cella di memoria con l’informazione cercata
2. La seconda serve per cercare velocemente gli impiegati e i progetti.

## Strutture Dati:

### 1. Tabella:

Una tabella occupa lo spazio minimo necessario a salvare n dati, e dato un indice restituisce il valore in tempo costante, perciò è la struttura migliore per il compito richiesto.

Quindi come vedremo meglio in seguito all’inizio dell’analisi viene costruita una tabella contenente i nomi degli impiegati, e per ogni impiegato verrà di volta in volta costruita una tabella con l’elenco dei progetti a cui questi possono partecipare.

### 2. Albero di ricerca binaria:

Sfruttiamo l’albero di ricerca binaria che avevamo costruito nell’analisi precedente, ricordiamo solo che è un albero che ha per nodo due argomenti

* Una stringa in cui è memorizzato il nome di un impiegato
* Un albero di ricerca binaria in cui sono memorizzati in ordine i progetti a cui può partecipare l’impiegato in esame

## Analisi Spaziale:

Il costo dell’albero è esattamente lo stesso dell’analisi spaziale precedente, cioè

Il costo della tabella in cui vengono salvati gli impiegati è esattamente uguale al numero di impiegati, mentre il costo della tabella in cui vengono salvati i progetti a cui possono partecipare gli impiegati è pari al numero di tali progetti, ma come vedremo questa tabella viene creata e liberata ogni volta che si cambia impiegato, così il costo spaziale di tale tabella è pari al massimo numero di progetti che può avere un singolo impiegato.

Riassumendo il costo spaziale sarà

## Analisi temporale:

Per prima cosa salviamo gli impiegati in una tabella, questo ha costo

1. Per ogni impiegato nella tabella, lo chiameremo **impiegato corrente** (in tutto ci sono ni impiegati)
   1. Creiamo una tabella in cui inseriamo i nomi dei progetti a cui potrebbe partecipare l’impiegato corrente (in tutto ci sono progetti)
   2. Per ogni impiegato che non è ancora stato controllato, lo chiameremo **impiegato da controllare** (ad ogni passaggio ne abbiamo ni-i)
      1. Per ogni progetto che fa parte dell’impiegato corrente, lo chiameremo **progetto corrente**
         1. Studiamo se il progetto corrente è nell’elenco dei progetti dell’impiegato da controllare; cerchiamo quindi nell’albero, l’impiegato da controllare (ci mettiamo un tempo ), una volta trovato scorriamo l’albero che contiene l’elenco di progetti (
            1. Se ciò è vero allora stampiamo il nome dell’impiegato corrente e il nome dell’impiegato da controllare, e passiamo subito ad analizzare un nuovo impiegato, senza vedere se ci sono altri progetti in comune.

Quindi il costo generale dell’algoritmo è

# Algoritmo greedy:

## Problema:

Vogliamo creare un algoritmo greedy per determinare il sottoinsieme di impiegati di peso massimo con competenze totalmente disgiunte, dove per peso di un impiegato si intende la somma dei livelli delle sue competenze.

Quindi per prima cosa dobbiamo trovare un modo per salvare per ogni impiegato l’Elenco degli impiegati che hanno almeno una competenza in comune con lui. Una volta fatto ciò potremo svolgere l’algoritmo greedy vero e proprio e stampare la soluzione.

## 1. Relazione impiegato-impiegato rispetto a competenze in comune:

Per risolvere questo problema usiamo esattamente lo stesso approccio che abbiamo utilizzato per calcolare la relazione impiegato-impiegato rispetto ai progetti, cioè costruiamo una tabella che contiene i nomi degli impiegati in ordine alfabetico, così che sia veloce e facile scorrerli, inoltre costruiamo un albero di ricerca binaria che ha come nodo una stringa che contiene il nome di un impiegato e un Elenco che contiene i nomi degli impiegati che sono in relazione con l’impiegato del nodo, questo Elenco è salvato a sua volta con albero di ricerca binaria. Questa scelta è stata fatta perché è più efficiente scorrere gli alberi rispetto ad altre strutture dati.

Quindi abbiamo

## Analisi spaziale:

Sia

* = numero di impiegati nell’elenco dell’impiegato i

Seguendo lo stesso ragionamento dell’analisi precedente otteniamo

## Analisi temporale:

Sia

* hei = altezza dell’albero che contiene l’elenco di impiegati dell’impiegato i

Seguendo lo stesso ragionamento dell’analisi precedente otteniamo

## 2. Algoritmo Greedy:

L’algoritmo greedy si divide in tre parti

1. Vengono caricati i dati e viene creata la struttura che conterrà la soluzione
2. Algoritmo greedy vero e proprio, utilizzando una versione modificata del problema dello zaino o knapsack
3. Stampa del risultato

Partiamo dal primo

## Caricamento dati e creazione soluzione

Per risolvere il problema di trovare il sottoinsieme disgiunto migliore ci serve carica i seguenti dati

* Una tabella di stringhe che contenga i nomi degli impiegati in ordine (l’abbiamo già dai passi precedenti)
* Un intero n che contenga la dimensione della tabella
* Un vettore di interi che contenga i pesi degli impiegati, cioè la somma dei livelli di tutte le competenze possedute da un impiegato (l’abbiamo chiamato phi)
* Un vettore di interi che contenga il numero di impiegati in relazione con ogni singolo impiegato (l’abbiamo chiamato v)

Quindi dobbiamo creare due vettori di dimensione 2n, allocando la memoria necessaria e scriverci sopra i dati, questo costa un tempo e uno spazio pari a

Per caricare la soluzione invece creiamo un vettore di dimensione n inizializzato tutto a FALSE, l’algoritmo greedy di volta in volta che un impiegato viene aggiunto alla soluzione cambierà l’indice di tale impiegato in TRUE, cosicchè alla fine la soluzione sia data dagli impiegati che hanno indice TRUE, inoltre per avere gli impiegati nell’ordine in cui gli abbiamo scelti, dobbiamo creare una tabella di interi in cui mettiamo gli indici degli impiegati della soluzione in ordine.

Anche qui il costo temporale e spaziale è di .

## Algoritmo greedy pseudo Knapsack

Finalmente arriviamo al cuore di questa sezione, cioè l’algoritmo greedy.

Ricapitoliamo un attimo cosa abbiamo e cosa vogliamo fare.

* Dati i nostri impiegati vogliamo creare il sottoinsieme migliore di impiegati totalmente disgiunti, dove per totalmente disgiunti si intende secondo la relazione impiegato-impiegato per competenze, e abbiamo appena visto che vogliamo scrivere la soluzione come un vettore di bouleani in cui le celle contenenti TRUE restituiscono l’indice in cui si trova l’impiegato che fa parte della soluzione.
* Per ottenere ciò abbiamo due vettori,
  + Il vettore pesi phi
  + Il vettore relazioni v
* Per valutare in che ordine inserire gli impiegati nella soluzione però non vogliamo prendere fra gli impiegati totalmente disgiunti, quelli di peso massimo in ordine, ma vogliamo prendere quelli per cui il rapporto fra il peso e il numero di relazioni sia il più grande possibile, cioè vogliamo che scegliere prima un impiegato che abbiamo un numero di relazioni il più basso possibile ma con il peso più alto possibile. Per far ciò introduciamo un nuovo vettore di double
  + Il vettore dei rapporti phi\_v
* Quindi sceglieremo di volta in volta l’impiegato col rapporto più grande e a parità di rapporto scegliamo quello con il peso più grande, ovviamente sempre tenendo conto che nella soluzione possono entrare solo impiegati disgiunti con chi è già stato inserito nella soluzione.

Per non dover cambiare ogni volt ai vettori creiamo dei vettori di indici sia per tener conto degli spostamenti del rapporto, sia degli spostamenti dei pesi.

Quindi l’algoritmo fa le seguenti cose

* Per prima cosa, con l’algoritmo HeapSort ordina l’indice dei pesi .
* Di volta in volta prende l’indice dell’impiegato con rapporto maggiore e tramite un heap indiretto lo inserisce nella prima posizione.
* Finchè il vettore che contiene il rapporto degli impiegati non è vuoto
  + Prende il primo elemento dell’indice (che abbiamo chiamato iMax) dei rapporti e controlla se può essere messo dentro la soluzione.  
    Se ciò è possibile
    - Inserisce l’indice nella soluzione, in due modi entrambi in tempo costante
      * Pone nella posizione iMax del vettore dei bouleani della soluzione TRUE.
      * Inserisce nella tabella di interi la posizione iMax, così da avere le soluzioni ordinate.
    - Somma all’intero che rappresenta la soluzione numerica il peso dell’indice iMax
* Toglie l’elemento di indice iMax dall’elenco di indici che controlla i rapporti, cioè phi\_v
* Aggiorna l’heap

### Aggiorna HeapindirettoRapporti:

Questa funzione è una normale funzione di heapindiretto, con un doppio confronto, cioè se il rapporto massimo in verità è massimale, allora si controlla il peso, questo viene fatto controllando se la posizione di iMax è maggiore o minore, nell’indice dei pesi che abbiamo ordinato prima con heapsort.

Il controllo della posizione nel caso di uguaglianza impiega in media un tempo .

Mentre l’heapindiretto impiega un tempo di

Allora l’aggiornamento di HeapIndirettoRapporti impiega un tempo pari a .

## Stampa

Infine stampiamo il risultato, cioè stampiamo

* La soluzione numerica che abbiamo trovato prima, in tempo costante
* Scorriamo la tabella delle soluzioni, che nel caso pessimo (cioè il caso in cui nessuno ha competenze comuni) ha un tempo , nella realtà è molto difficile che questo accada e quindi sarà di meno, ma non può mai essere di più

Per concludere quindi tutto l’algoritmo greedy, dal caricamento alla stampa consuma un tempo

Cioè in effetti dipende solo dall’effettivo algoritmo e non dal caricamento e dalla stampa.