Progetto Algoritmi e Strutture Dati

Jacopo Peroni

Giugno, 2023

Indice

1 Problema

L'algoritmo verte a risolvere una necessità di una pizzeria, trasformare delle volotà espresse dai clienti nei confronti di un insieme di ingredienti, in informazioni utili dal punto di vista gestionale.
I dati forniri sono:

- Elenco degli ingredienti (il cui numero ai fini dei calcoli della complessità chiameremo m).
- Elenco dei clienti (il cui numero ai fini dei calcoli della complessità chiameremo n) ove, per ogni cliente, vengono indicate le sue preferenze espresse come RICHIESTI, GRADITI e ESCLUSI.

Le richieste della pizzeria sono:

- Stampare gli ingredienti in ordine di popolarità decrescente.
- Stampare i clienti in ordine di grado di esigenza crescente.
- Stampare il numero di ingredienti che vengono esclusi da un dato numero di clienti.
- Stampare l'elenco delle coppie di clienti incompatibili.
- Stampare la stima per difetto del numero di pizze, comprensiva anche dell'elenco dei clienti che causano questo minimo di pizze.
- Stampare la lista delle pizze come elenco degli ingredienti che le compongono e dei clienti che soddisfano.

La popolarità è data (nell'ordine) dal minor numero di esclusioni, dal maggior numero di richiesti e poi dal maggior numero di graditi (le parità sono decise per ordine alfabetico).
L'esigenza di un cliente è data dal numero di esclusioni e richieste che ha posto.
L'incompatibilità è data dalla contemporanea esclusione e richiesta dello stesso ingrediente da parte di due clienti.
Per la stima per difetto e per la lista delle pizze ci si rifà alle regole degli algoritmi *greedy* descritti nel file dell'esercizio.

2 Struttura del Codice

Il codice si basa sull'utilizzo di **5 librerie** per le strutture dati e per gli algoritmi di heapsort, non si tratta dei codici visti/codificati a lezione ma di una ricodifica adattata alla necessità del problema.
Il problema viene risolto dal file **main.c** che è a sua volta costituito da 17 funzioni (più il main) divise in chunk a seconda del sottoproblema a cui sono riferite (questo semplifica la lettura partizionando in 7 sezioni: la prima di lettura dai file e le sei relative ai sottoproblemi).
Sarebbe stato sicuramente possibile andare a trasportare queste funzioni nelle librerie .h relative ma avrebbe reso il progetto pieno di ulteriori file (considernado il .h ed il .c per ogni sottoproblema avremmo altri 12 file) con la complicazione di librerie che si chiamano a vicenda per via di funzioni utilizzate in più sottoproblemi.
L'ordine degli eventi nel codice è dato dalla lettura dei dati dai file nelle strutture date preposte seguito poi dal calcolo, e dalla stampa, delle soluzioni dei 6 sottoproblemi.
In questa relazione tratterò prima le librerie utilizzate, in ordine alfabetico, e poi il file main.c. L'ordine in cui tratto le funzioni è esattamente quello in cui compaiono nel file relativo.

3 Approccio al problema

Il mio approccio al problema è stato quello più attinente all'analisi dati, ovvero quello a *Database*, infatti la mia soluzione si basa su due tabelle statiche, che sono, l'elenco dei clienti e degli ingredienti e su una matrice, anch'essa statica, chiamata **preferenze** di grandezza nxm.
Quest'ultima presenta come entrate: la preferenza espressa da un cliente (riga) su un determinato ingrediente (colonna) dove il legame tra un indice (riga o colonna) ed il relativo nome è dato dalle relative tabella statiche. Il concetto è che l'indice nelle tabelle è una *primary key* mentre nella matrice fa da *foreign key*.
La preferenza è espressa tramite un intero:

- RICHIESTO: 2
- ESCLUSO: -2
- GRADITO: 1
- NESSUNA RELAZIONE: 0

Chiaramente ogni sottoproblema presenta poi strutture aggiuntive utili per la richiesta specifica.

4 Note sull'analisi

L'analisi della mia soluzione può essere divisa in due macro-blocchi: il primo per le librerie ed il secondo per il problema vero e proprio.
Visto che le librerie sono molto simili a quelle viste a lezione (anche se non totalmente uguali come già detto) e non sono il fulcro della risoluzione del problema, propongo una tabella in cui do le complessità di ognuna di esse. La descrizione delle singole funzioni verrà data tamite commento direttamente nel codice.
In generale nelle analisi di costo temporale ho trascurato alcune operazioni che avvengono in tempo costante (in particolare le definizioni delle variabili statiche e alcuni cambiamenti di valore di singole variabili).
Per quanto riguarda la complessità spaziale, ho trascurato l'allocazione delle variabili statiche, che occupano una quantità di memoria prestabilita, e il passaggio dei parametri a funzioni.

5 Librerie

5.1 heapsort.c

Sono presenti nel codice due varianti, quasi identiche, del heapsort.
Nella prima (fatta da **aggiornaheap**, **creaheap**, **heapSortCustom**) effettuo un heapsort su una matrice, quindi definisco una riga su cui verrà applicato l'algoritmo di ordinamento in questione, imponendo l'obbligo di scambiare tutte le colonne come entità unica.
Quindi se l'ordinamento avviene su una riga anche le altre righe dovranna effettuare gli stessi scambi tramite la funzione **invertiColonneMatrice**.
Chiamo m il numero di colonne della matrice e n il numero di righe.
La seconda (fatta da **aggiornaheapLex**, **creaheapLex**, **heapSortCustomLex**) variante è pressoche identica, con l'unica differenza nel tipo di confronto che viene effettuato trami lessicografico tra i nomi nella tabella **nomi**.

Nome	Complessità Temporale	Complessità Spaziale
aggiornaheap	$O(nlogm)$	$O(1)$
creaheap	$O(nmlogm)$	$O(1)$
heapSortCustom	$O(nmlogm)$	$O(1)$
aggiornaheapLex	$O(nlogm)$	$O(1)$
creaheapLex	$O(nmlogm)$	$O(1)$
heapSortCustomLex	$O(nmlogn)$	$O(1)$

5.2 liste.c

Le strutture dati introdotte in questa libreria sono:

- Un nodo ad interi con puntatore al blocco successivo e precedente, vengono anche definiti puntatori al nodo con due nomi *pizza* e *ingrediente* per pura ragione estetica e di comprensione del codice. Le liste saranno l'elenco degli ingredienti, dove il numero intero contenuto nei nodi sarà la *foreign key* alla tabella ingredienti.
- Viene anche introdotta una struttura dati **menu** con un puntatore a pizza (utilizzato per definire un array dinamico di pizze, ovvero puntatori a nodi iniziali di liste, che utilizzeremo come elenco di ingredienti) ed un intero che definisce la lunghezza del vettore dinamico da istanziare.

Di conseguenza le funzioni nella libreria sono alcune per la gestione del menu (**creaMenuLista**, **nuovaPizzaInMenu**, **leggiPizzaDaMenu**, **cambiaPizza** e **cancellaMenu**) e alcune per la gestione della lista (tutte le altre).
Chiamo n il numero di elementi della lista.

Nome	Complessità Temporale	Complessità Spaziale
creaLista	$\Theta(1)$	$O(1)$
creaMenuLista	$\Theta(1)$	$O(1)$
nuovaPizzaInMenu	$\Theta(1)$	$O(1)$
leggiPizzaDaMenu	$\Theta(1)$	$O(1)$
cambiaPizza	$\Theta(1)$	$O(1)$
recuperaLunghezzaLista	$\Theta(n)$	$O(1)$
nextIngredienteLista	$\Theta(1)$	$O(1)$
inserisciElementoLista	$\Theta(1)$	$O(1)$
leggiElementoLista	$\Theta(1)$	$O(1)$
eliminaElementoLista	$\Theta(1)$	$O(1)$
inserisciNodoFondoLista	$\Theta(1)$	$O(1)$
creaCopiaLista	$\Theta(n)$	$O(1)$
indiceDatoContenutoInLista	$O(n)$	$O(1)$
cancellaLista	$\Theta(1)$	$O(1)$
cancellaMenu	$O(1)$	$O(1)$

5.3 matrice.c

Ipotizzando di poter trattare realloc come $\Theta(n)$ con n numero elementi nel vettore.
Sia n il numero di righe e m il numero di colonne, concorde con la notazione di clienti ed ingredienti.
Per la funzione *listaInMatrice* k è la lunghezza della lista.

Nome	Complessità Temporale	Complessità Spaziale
inizializzaMatrice	$\Theta(nm)$	$\Theta(nm)$
numeroRigheMatrice	$\Theta(1)$	$O(1)$
numeroColonneMatrice	$\Theta(1)$	$O(1)$
recuperaMatrice	$\Theta(1)$	$O(1)$
aggiungiRigaVuotaMatrice	$\Theta(n)$	$O(m)$
inserimentoRigaMatrice	$\Theta(m)$	$O(1)$
inserimentoElementoMatrice	$\Theta(1)$	$O(1)$
leggiValoreMatrice	$\Theta(1)$	$O(1)$
aggiungiElementoMatrice	$\Theta(1)$	$O(1)$
numeroColonneNonVuote	$\Theta(nm)$	$O(n)$
invertiColonneMatrice	$\Theta(n)$	$O(1)$
copiaColonnePorzioneMatrice	$O(nm)$	$O(1)$
copiaColonnaInRiga	$O(n)$	$O(n)$
confrontaRigaColonna	$O(n)$	$O(1)$
aggiuntaColonnaMatrice	$\Theta(nm)$	$O(n)$
indiceMassimoRispettoARigaMatrice	$O(m)$	$O(1)$
indiceMinimoRispettoARigaMatrice	$O(m)$	$O(1)$
creaCopiaMatrice	$\Theta(nm)$	$O(1)$
listaInMatrice	$\Theta(k)$	$O(k)$
cancellaMatrice	$\Theta(n)$	$O(1)$

5.4 matriceSimmetrica.c

Ipotizzando di poter trattare calloc come $\Theta(n)$ con n numero elementi nel vettore.
Sia la dimensione n.

Nome	Complessità Temporale	Complessità Spaziale
inizializzaMatriceSimmetrica	$\Theta(n(n+1)/2)$	$O(n(n+1)/2)$
inserisciValoreMatriceSimmetrica	$\Theta(1)$	$O(1)$
leggiValoreMatriceSimmetrica	$\Theta(1)$	$O(1)$
leggiGrandezzaMatriceSimmetrica	$\Theta(1)$	$O(1)$
cancellaMatriceSimmetrica	$\Theta(1)$	$O(1)$

5.5 tabella.c

Considerando che la lunghezza massima dei nomi dei clienti e degli ingredienti ci è fornito la fisso come costante pari a 50.
Sia n il numero di elementi nella tabella (essendo un vettore di stringhe considero il numero di stringhe).

Nome	Complessità Temporale	Complessità Spaziale
inizializzaTabella	$O(1)$	$O(1)$
inserimentoElementoTabella	$\Theta(n)$	$O(1)$
trovaIndiceDaContenutoTabella	$O(n)$	$O(1)$
leggiValoreTabella	$\Theta(1)$	$O(1)$
leggiLunghezza	$\Theta(1)$	$O(1)$
cancellaTabella	$\Theta(n)$	$O(1)$

6 main.c

Come librerie vengono utilizzate *stdio.h*, *stdlib.h* e *string.h* più tutte quelle definite sopra.
Sono presenti 5 costanti, la prima (NUMERO_PREFERENZE) determina quanti tipi di preferenze possono essere espresse dai clienti, e le altre 4 sono i valori numerici che definiscono il tipo di preferenza espressa da un cliente nei confronti di un ingrediente (vedi Approccio al Problema).

L'algoritmo è diviso in 6 sottoproblemi delineati anche dalla suddivisione dei prototipi delle funzioni:

- Lettura dati da file e creazione strutture dati (**interpretaLineaComando**, **letturaIngredientiDaFile** e **letturaClientiDaFile**)
- Calcolo e stampa della classifica di popolarità degli ingredienti (**inizializzaClassificaPopolarita**, **ordinaClassifica** e **calcolaOrdinePopolarita**)
- Calcolo e stampa della classifica per esigenza dei clienti (**inizializzaClassificaEsigenza** e **calcolaOrdineEsigenza**)
- Calcolo e stampa numero ingredienti esclusi per dato numero di clienti (**inizializzaClassificaIngrEsclusi** e **ingredientiEsclusiPerNumeroClienti**)
- Calcolo e stampa delle coppie incompatibili (**calcoloCoppieIncompatibili**)
- Calcolo e stampa della stima per difetto del numero di pizze e dell'elenco di clienti che generano la stima calcolato tramite greedy (**calcoloNumeroIncompatibiliPerCliente** e **calcoloStimaInfNumeroPizza**)
- Calcolo e stampa della lista delle pizze nel menu calcolato tramite greedy (**aggiornaListaClientiSoddisfatti**, **modificaPreferenzeIngrediente**, **creazionePizza** e **calcoloStimaMenuPizza**)

6.1 main

La funzione main fa solo da pipeline, quindi definisce le variabili che verranno usate per tutto il codice, chiama le funzioni necessarie per risolvere i vari problemi ed infine dealloca le variabili.

6.1.1 Variabili e Strutture Dati

Le variabili definite sono:

- **nomeFileIngredienti** e nomeFileClienti *stringhe* che conterranno i nomi dei file da cui leggere i dati sugli ingredienti, sui clienti e sulle preferenze.
- **ingredienti** e clienti *tabelle* che conterranno i vari nomi degli ingredienti e dei clienti.
- **preferenze** *matrice* che conterrà le preferenze espresse da ogni cliente su ogni ingrediente (anche la non espressione di una preferenza è salvata), le righe saranno i clienti e le colonne gli ingredienti.
- **incompatibili** *matriceSimmetrica* che conterrà 1 se due clienti sono incompatibili e 0 se sono compatibili, viene scelta una matrice simmetrica per risparmiare merità dello spazio, infatti partendo dall'idea di confrontare ogni cliente con ogni altro cliente è facile vedere che ponendo l'incompatibilità in una matrice essa sarà simmetrica.

6.1.2 Algoritmo

Per prima cosa viene lanciata la funzione **interpretaLineaComando** con il compito di leggere da linea di comando il percorso dei file degli ingredienti e dei clienti con le preferenze e di salvare questi percorsi nelle variabili *nomeFileIngredienti* e *nomeFileClienti*.
Vengono poi chiamate le funzioni **inizializzaTabella** e **letturaIngredientiDaFile** con cui viene allocata la tabella *ingredienti* e vengono salvati i nomi di quest'ultimi letti dal file prima recepito da terminale.
Vengono poi chiamate le funzioni **inizializzaTabella**, **inizializzaMatrice** (con anche **leggiLunghezza**) e **letturaClientiDaFile** con cui viene allocata la tabella *clienti* e la matrice *preferenze* e vengono salvati i nomi dei clienti e le loro preferenze espresse lette dal file prima recepito da terminale.
Viene poi chiamata **calcolaOrdinePopolarita** che data la matrice preferenze e la tabella ingredienti calcola e stampa la classifica degli ingredienti per popolarità dicrescente.
Successivamente viene chiamata **calcolaOrdineEsigenza** che data la matrice preferenze e la tabella clienti calcola e stampa la classifica dei clienti in ordine di grado di esigenza.
Viene poi chiamata **ingredientiEsclusiPerNumeroClienti** che data la matrice preferenza calcola e stampa il numero di ingredienti esclusi da un dato numero di clienti (partendo da 0).
Viene poi chiamata **calcoloCoppieIncompatibili** che prende la matrice simmetrica incompatibili, la matrice preferenze e la tabella cienti per calcolare e salvare in incompatibili e incompatibilità e per stampare le coppie incompatibili.
Infine sono presenti gli algoritmi greedy: **calcoloStimaInfNumeroPizza** che data la matrice incompatibili e la tabella clienti stampa il numero minimo di pizze necessarie e l'elenco dei clienti che generano questo minimo, e **calcoloStimaMenuPizza** che data la matrice preferenze e le tabelle ingredienti e clienti calcola e stampa la lista delle pizze per un menu che soddisfi tutti i clienti, la stampa avviene tramite elenco degli ingredienti e elenco dei clienti soddisfatti.
Come ultimo passaggio viene deallocata la memoria delle variabili *ingredienti*, *clienti*, *preferenze* e *incompatibili*.

6.1.3 Complessità

La complessità spaziale di interpretaLineaComando è $\Theta(1)$.
La complessità spaziale di inizializzaTabella è $\Theta(1)$.
La complessità spaziale di letturaIngredientiDaFile è $\Theta(m)$.
La complessità spaziale di inizializzaTabella è $\Theta(1)$.
La complessità spaziale di leggiLunghezza è $\Theta(1)$.
La complessità spaziale di inizializzaMatrice è $\Theta(m)$.
La complessità spaziale di letturaClientiDaFile è $\Theta(nm)$.
La complessità spaziale di calcolaOrdinePopolarita è $\Theta(m)$.
La complessità spaziale di calcolaOrdineEsigenza è $\Theta(n)$.
La complessità spaziale di ingredientiEsclusiPerNumeroClienti è $\Theta(m)$.
La complessità spaziale di calcoloCoppieIncompatibili è $\Theta(n^2)$.
La complessità spaziale di calcoloStimaInfNumeroPizza è $O(n)$.
La complessità spaziale di calcoloStimaMenuPizza è $O(nm)$.
La complessità spaziale di cancellaTabella è $\Theta(1)$.
La complessità spaziale di cancellaTabella è $\Theta(1)$.
La complessità spaziale di cancellaMatrice è $\Theta(1)$.
La complessità spaziale di cancellaMatriceSimmetrica è $\Theta(1)$.
La complessità spaziale totale è quindi $O(m + m + nm + m + n + m + n^2 + n + nm) = O(nm + n^2)$.

La complessità temporale di interpretaLineaComando è $\Theta(1)$.

La complessità temporale di inizializzaTabella è $\Theta(1)$.
La complessità temporale di letturaIngredientiDaFile è $\Theta(m^2)$.
La complessità temporale di inizializzaTabella è $\Theta(1)$.
La complessità temporale di leggiLunghezza è $\Theta(1)$.
La complessità temporale di inizializzaMatrice è $\Theta(nm)$.
La complessità temporale di letturaClientiDaFile è $O(n^2m)$.
La complessità temporale di calcolaOrdinePopolarita è $\Theta(nm + mlogm)$ nel caso migliore e $\Theta(nm + m^2logm)$ nel caso peggiore, ovvero $O(nm + m^2logm)$.
La complessità temporale di calcolaOrdineEsigenza è $\Theta(nm+nlogn)$ nel caso migliore e $\Theta(nm+n^2logn)$ nel caso peggiore, ovvero $O(nm + n^2logn)$.
La complessità temporale di ingredientiEsclusiPerNumeroClienti è $\Theta(nm + mlogm)$.
La complessità temporale di calcoloCoppieIncompatibili è $\Theta(n^2m)$ nel caso migliore e $\Theta(n^2m + n^3)$ nel caso peggiore, ovvero $O(n^2m + n^3)$.
La complessità temporale di calcoloStimaInfNumeroPizza è $O(n^2logn)$ nel caso migliore e $O(n^3)$ nel caso peggiore, la tratto quindi come $O(n^3)$.
La complessità temporale di calcoloStimaMenuPizza è $O(nm^2 + n^2logn)$ nel caso migliroe e $O(n^3m + n^2m^2)$ nel caso peggiore, la tratto quindi come $O(n^3m + n^2m^2)$.
La complessità temporale di cancellaTabella è $\Theta(m)$.
La complessità temporale di cancellaTabella è $\Theta(n)$.
La complessità temporale di cancellaMatrice è $\Theta(n)$.
La complessità temporale di cancellaMatriceSimmetrica è $\Theta(1)$.
La complessità temporale totale è quindi $O(m^2 + nm + n^2m + nm + m^2logm + nm + n^2logn + nm + mlogm + n^2m + n^3 + n^3 + n^3m + n^2m^2 + m + n + n) = O(m^2logm + n^3m + n^2m^2)$.

6.2 interpretaLineaComando

6.2.1 Algoritmo

Controlla che il numero di argomenti passati da terminale siano 3, in caso contrario lancia errore e termina l'esecuzione.
Copia il 2° argomento del terminale in nomeFileIngredienti e il 3° argomento in nomeFileClienti.

6.2.2 Complessità

Non sono definite variabili quindi la complessità spaziale è $\Theta(1)$.

Il blocco if ha complessità temporale $\Theta(1)$.
Copiare le stringhe ha complessità temporale $\Theta(1)$.
La complessità temporale è $\Theta(1)$.

6.3 letturaIngredientiDaFile

6.3.1 Variabili e Strutture Dati

Le variabili defintie solo *file* FILE* per aprie il file contentente gli ingredienti, *numeroIngredienti* INT per salvare il numero degli ingredienti, *i* INT come iteratore per il ciclo e *ingrediente* vettore CHAR per salvare il nome dell'ingrediente letto prima di inserirlo nella tabella.

6.3.2 Algoritmo

Viene aperto il file contentente gli ingrediente in lettura, se l'apertura non va a buon fine viene lanciato un errore e terminata l'esecuzione.

Viene letta la prima riga che conterrà il numero di ingredienti che si andrà a leggere, questo numero è salvato in una variabile.
In un ciclo con un numero di iterazioni pari al numero di ingredienti viene letta riga per riga il file, salvando ogni volta il contenuto in una variabile e poi chiamando la funzione *inserimentoElementoTabella* per inserire nella tabella ingredienti il nuovo ingrediente letto.
Infine chiudo il file.

6.3.3 Complessità

Chiamo m volte inserimentoElementoTabella quindi la complessità spaziale è $\Theta(m)$.

L'apertura del file ed il blocco if di controlo hanno complessità temporale $\Theta(1)$.
La lettura del numero di ingredienti ha complessità temporale $\Theta(1)$.
Il ciclo for ha complessità temporale $\Theta(m^2)$ essendo la funzione inserimentoElementoTabella $\Theta(m)$.
Di conseguenza la complessità temporale totale è $\Theta(m^2)$.

6.4 letturaClientiDaFile

6.4.1 Variabili e Strutture Dati

Le variabili definite sono *file* FILE* per aprire il file contenente i clienti e le loro preferenze, *numeroClienti* INT per salvare il numero di clienti, i, j e k INT iteratori per i cicli, *cliente* vettore di CHAR dove salvare il nome del cliente quando letto, *numeroPreferenzeEspresse* INT per salvare quante preferenze di un determinato tipo sono state espresse da un cliente, *preferenza* vettore di CHAR dove salvare il nome dell'ingrediente su cui il cliente sta esprimento la preferenza, *indicePreferenza* INT primaryKey della tabella ingredienti da usare come foreignKey in preferenze, *elencoPreferenze* INT* dove mettere le preferenze di un cliente prima di salvarle nella matrice preferenze e *listaPreferenzePossibili* vettore di INT dove metto i valori delle costanti che indicano le preferenze espresse.

6.4.2 Algoritmo

Alloco un vettore di interi per contenere tutte le preferenze, una per ingrediente, il numero ingredienti lo prendo con leggiLunghezza. Di conseguenza effettuo un controllo se l'allocazione è andata a buon fine se no lancio un errore e termina l'esecuzione.
Inizializzo listaPreferenzePossibili con le 3 possibili preferenze espresse dai clienti.
Apro il file con i clienti e le preferenze in lettura, in caso di mancata apertura lancio errore e termina esecuzione.
Viene letto il numero di clienti e salvato in una variabile.
Inizio un ciclo dove per ogni cliente leggo il nome, lo salvo in una variabile e poi lo aggiungo alla tabella clienti, poi effettuo un ciclo per inizializzare l'elenco delle preferenze di questo cliente mettendo di base NESSUNA_RELAZIONE, poi ciclo sul numero di preferenze possibili (ognuna ha una riga con gli ingredienti che ricadono in essa). Per ogni riga leggo il numero di preferenze e lo salvo in una variabile, per ogni preferenza leggo il nome dell'ingrediente, ne trovo l'indice nella tabella degli ingredienti ed inserisco nell'elenco delle preferenze (in corrispondenza di quell'indice) il tipo di preferenza espressa.
Se non sono al primo giro del ciclo aggiungo una riga vuota alla matrice preferenze.
Inserisco poi la riga con l'elenco delle preferenze nella matrice preferenze.
Infine dealloco la variabile elencoPreferenze e chiudo il file.

6.4.3 Complessità

La variabile *elencoPreferenze* allocata occupa uno spazio dell'ordine $\Theta(m)$, tutte le altre variabili occupano spazio costante.
La funzione aggiungiRigaVuotaMatrice occupa uno spazio dell'ordine $\Theta(m)$ e viene chiamata $n-1$ volte. La complessità spaziale della funzione è $\Theta(m+(n-1)m) = \Theta(nm)$.

L'allocazione ed il blocco if relativo a elencoPreferenze ha complessità temporale $\Theta(1)$.
L'apertura del file ed il blocco if relativo ha complessità temporale $\Theta(1)$.
Il primo ciclo for effettua n cicli, la funzione inserimentoElementoTabella ha complessità $\Theta(n)$, il secondo ciclo for effettua m cicli quindi ha complessità $\Theta(m)$, il terzo ciclo for è più particolare.
Effettua 3 cicli dove per ognuno effettua un ciclo for lungo quanto il numero di preferenze espresse, esso può essere al più pari al numero di ingredienti, ovvero n, se però una preferenza è fatta di n ingredienti allora le altre 2 sono vuote, in quanto un cliente non può esprimere differenti preferenze sullo stesso ingrediente.
In questo quarto ciclo for viene chiamata la funzione trovaIndiceDaContenutoTabella che ha complessità pari a $\Theta(m)$.
Chiuso il terzo for, $n-1$ volte viene aggiunta una riga con aggiungiRigaVuotaMatrice che ha complessità pari a $\Theta(n)$.
Viene poi inserito il vettore elencoPreferenze in preferenze con inserimentoRigaMatrice che ha complessità pari a $\Theta(m)$.
La complessità temporale finale è $O(n(n+m+3nm+n+m)-n) = O(2n^2+2nm+3n^2m-n) = O(n^2m)$.

6.5 inizializzaClassificaPopolarita

6.5.1 Variabili e Strutture Dati

Le variabili definite sono *i*, *j* INT iterabili per i cicli, *elementoMatrice* INT dove salvo l'elemento letto dalla matrice preferenze e *numeroRighePreferenze* INT numero di clienti.

6.5.2 Algoritmo - Idea

L'idea è ottenere una matrice 4 x numero di ingredienti dove nella prima riga ho l'indice dell'ingrediente, nella seconda riga quante volte quell'ingrediente è ESCLUSO, nella terza quante volte è RICHIESTO e nella quarta quante volte è GRADITO.

6.5.3 Algoritmo - Tecnica

Se ci sono 0 ingredienti lancia errore e termina l'esecuzione.
Inizializzo una matrice chiamata classifica per avere 4 righe e numeriIngredientiNominati colonne, le 4 righe sono 1 per le foreign key e 3 per il numero di esclusioni, richiesti e gradimenti per ogni ingrediente.
Effettuo un ciclo for di numeroIngredientiNominati cicli in cui inserisco nella prima riga in posizione j-esima il j-esimo elemento del vettore ingredinetiNominati, così da avere in prima riga gli le foreignKey della tabella ingredienti.
Mi salvo il numero di righe della matrice preferenze, ovvero il numero di clienti.
Effettuo un ciclo di numeroIngredientiNominati istanze, per ogni ciclo effettuo un ulteriore for di numeroRighePreferenze cicli dove leggo la preferenza del j-esimo cliente rispetto al i-esimo ingrediente

e se è ESCLUSO aggiungo 1 nella riga 1, se è RICHIESTO sottraggo 1 nella riga 2 e se è GRADITO sottraggo 1 nella riga 3 della matrice classifica. L'ordine delle righe è dato dalla definizione di popolarità che vede l'importanza delle preferneze in quest'ordine. L'utilizzo di somme per ESCLUSO e sottrazioni per RICHIESTO e GRADITO serve per permettere, tramite ordinamento crescente, di ottenere in meno escluso come primo elemento ed il più richiesto/gradito come primo elemento.

6.5.4 Complessità

Le variabili definite nella funzione hanno grandeza costante.
La funzione inizializzaMatrice ha complessità spaziale $\Theta(4m) = \Theta(m)$.
Le altre funzioni chiamate hanno tutte complessità spaziale costante.
La complessità spaziale della funzione è $\Theta(m)$.

Il primo blocco if ha complessità temporale costante.
La funzione inizializzaMatrice ha complessità temporale $\Theta(nm)$.
Il ciclo for successivo chiama per n volte la funzione inserimentoElementoMatrice che ha complessità lineare, quindi il ciclo ha complessità $\Theta(n)$.
La lettura del numeroRigheMatrice ha complessità costante.
Il blocco successivo di for cicla per n volte ed ha all'interno un ciclo for che cicla m volte.
Quest'ultimo chiama la funzione leggiValoreMatrice che è costante e nel peggiore dei casi, in cui ogni cliente esprime una preferenza tra le 3 possibili su tutti gli ingrendienti, ogni volta chiama aggiungiElementoMatrice che ha complessità costante.
Quindi la complessità temporale di questa funzione è $\Theta(nm + n + nm) = \Theta(nm)$.

6.6 ordinaClassifica

Questa funzione è utilizzata non solo nel primo sottoproblema ma anche nel primo e nel secondo greedy.

6.6.1 Variabili e Strutture Dati

Le variabili definite sono i, j e k INT iterabili per i cicli e *tempVal* RIGA utilizzata per copiare porzioni di colonne della matrice classifica.

6.6.2 Algoritmo - Idea

L'idea è ordinare una classifica di h righe dove la riga 0 non va toccata in quanto fatta di indici. La classifica viene quindi ordinata in maniera crescente secondo i valori nella riga 1.
Ora per ogni gruppo che ha lo stesso valore nella riga 1, ordino rispetto alla riga 2.
Questo processo continua finché non compio numeroLivelliOrdinamento ordinamenti.
Per l'ultimo giro l'ordinamento avviene tramite l'ordine lessicografico dei nomi relativi alla tabella tabellaNomi, usando gli indici della riga 0.

6.6.3 Algoritmo - Tecnica

Effettuo un controllo se numeroLivelliOrdinamento è minore di 0 o maggiore di 4 in quanto sono valori non ammissibili per questo problema, in caso stampa errore e termina esecuzione.
Parte poi un ciclo che compie numeroLivelliOrdinamento cicli, al massimo 4, l'indice del ciclo corrente è k.
Faccio partire i e j da 0.

i sarà l'indice del punto da cui parte il blocco da ordinare, j sarà un indice movente per tutte le colonne di classifica. Copio, a partire dalla riga 1, k righe della colonna j dalla matrice classifica alla riga tempVal. Copiare k righe serve per tenere da conto gli ordinamenti fatti nelle righe precedenti. Infatti ogni volta che i numeri nelle righe sopra cambiano so che non devo effettuare l'ordine rispetto alle riga corrente. Se la quantità di colonne per cui le righe sopra sono rimasti costanti è pari a 1 non ordino, se è maggiore di 1 dovrò ordinare la sottomatrice data dall'intervallo in cui le righe sopra erano costanti.
Inizio un ciclo while che ha come condizione di uscita j maggiore o uguale al numeroElementiClassifica.
Confronto tempVal con la colonna j della classifica (chiaramente k rimane costante per tutto il while, così le lunghezze per il confronto sono concordi), se sono diversi e se la distanza tra i e j è maggiore di 1 (ovvero se il blocco da ordinare è fatto da più di una colonna), allora, inizializzo una matrice con numeroLivelliOrdinamento righe e con tante colonne quanto la lunghezza del blocco da ordinare.
Copio il contenuto del blocco tra i e j nella matrice appena creata tempM.
Se NON sono all'ultimo livello di ordinamento effettuo un heapSortCustom sulla tempM con riga per i confronto l'ultima.
Se sono all'ultimo livello di ordinamento chiamo heapSortCustomLex che confronta lessicografico rispetto a tabellaNomi.
Quindi ricopio il contenuto di tempM (grande uguale, ma ordinata) in classifica nella sottomatrice da cui era stata copiata prima, così ottengo la matrice classifica con la sottomatrice ordinata.
Dopo l'ordinamento o se la larghezza del blocco era 1 sposto l'indice i su j così da inziare a considerare un nuovo blocco da ordinare. Di conseguenza copio il l'indicatore del blocco (la colonna j) in tempVal andando a sovrascrivere quella precedente.
Incremento j di 1.
Finito il ciclo while (ultima colonna da controllare) effettuo l'ordine descritto precedentemente in qualsiasi caso, perché può capitare che la fine dell'ultimo blocco da ordinare non sia definita da un cambio nelle righe precedenti ma dalla effettiva fine della matrice classifica.

6.6.4 Complessità

La complessità spaziale non è influenzata dalla funzione copiaColonnaInRiga che agisce su tempVal in quanto al più genera un vettore di 4 interi, quindi costante.
Però è influenzata dalla funzione inizializzaMatrice che agisce su tempM copiando una matrice di al più 4 righe ma con un numero di colonne che cresce come in numeroElementiClassifica (che denominiamo m), quindi ha una complessità spaziale dell'ordine $\Theta(4m) = \Theta(m)$.
Essendo che sovrascrivo sempre tempVal e tempM la complessità spaziale generale è $\Theta(m)$.

Per la complessità temporale il blocco if iniziale è costante.
Il primo ciclo for può fare al più 4 iterazioni quindi è costante.
La funzione copiaColonnaInRiga ha complessità temporale lineare con l'altezza della colonna, che è al più 4 quindi lineare.
Il ciclo while effettua m iterazioni.
La funzione confrontaRigaColonna ha complessità temporale lineare con l'altezza della colonna, che è al più 4 quindi lineare.
La condizione nel blocco if successivo mi permettere di distinguere il caso migliore dal caso peggiore, in caso fosse già ordinato senza blocchi in cui più ingredienti hanno lo stesso numero di esclusioni e quindi vanno ordinati per richieste, ho la condizione dell'if che non viene mai soddisfatta e quindi il blocco ha complessità lineare. Nel caso peggiore, ovvero quando il blocco da ordinare (da i a j) è lungo quanto tutta la matrice o quando viene ripetuto l'ordinamento su più blocchi che partizionano la matrice, calcolo la complessità.
La funzione inizializzaMatrice ha complessità $\Theta(4m) = \Theta(m)$.

La funzione copiaColonnePorzioneMatrice ha complessità $\Theta(4m) = \Theta(m)$.
Il blocco if else chiama o heapSortCustom o heapSortCustomLex che hanno la stessa complessità pari a $\Theta(4mlogm) = \Theta(mlogm)$.
La funzione copiaColonnaInRiga ha complessità lineare.
Finito il ciclo while effettuo un ultimo ordinamento quindi:
La funzione inizializzaMatrice ha complessità $\Theta(4m) = \Theta(m)$.
La funzione copiaColonnePorzioneMatrice ha complessità $\Theta(4m) = \Theta(m)$.
Il blocco if else chiama o heapSortCustom o heapSortCustomLex che hanno la stessa complessità pari a $\Theta(4mlogm) = \Theta(mlogm)$.
La funzione copiaColonnaInRiga ha complessità lineare.
La complessità temporale nel caso migliore è $\Theta(m + m + m + mlogm) = \Theta(mlogm)$.
La complessità temporale nel caso peggiore è $\Theta(m(m+m+mlogm)+m+m+mlogm) = \Theta(m^2logm)$.

6.7 calcolaOrdinePopolarita

6.7.1 Variabili e Strutture Dati

Le variabili definite sono *classifica* MATRICE utilizzata per ordinare tramite la funzione ordinaClassifica, *ingredientiNominati* RIGA vettore di idici degli ingredienti nominati almeno una volta, *numeroIngredientiNominati* INT numeo di ingredienti nominati almeno una volta e i, j INT iterabili per cicli.

6.7.2 Algoritmo - Idea

L'idea è utilizzare le due funzioni precedenti per costruire una matrice che contenga gli indici degli ingredienti nominati almeno una volta, il numero di esclusioni, il numero di richieste ed il numero di gradimenti. Successivamente ordinarla e poi stampare la classifica.

6.7.3 Algoritmo - Tecnica

Recupero il numero di ingredienti che cono stati nominati almeno una volta da un cliente e salvo il loro indici in un vettore di interi tramite la funzione numeroColonneNonVuote.
Stampo il numero di ingredienti seguito dalla parola ingredienti.
Costruisco la matrice 4 x numero di ingredienti nominati tramite la funzione inizializzaClassificaPopolarita e la salvo nella matrice classifica.
Ordino la classifica sencondo le regole definite nel PDF del progetto tramite la funzione ordinaClassifica.
Stampo poi come da richiesta, per ogni ingrediente nominato, nell'ordine decrescente della classifica: il nome dell'ingrediente (preso come la stringa nella tabella ingredienti con indice l'elemento nella riga 0 della colonna in lettura di classifica), il numero di esclusioni, il numero di richieste ed il numero di gradimenti.

6.7.4 Complessità

Considero che numeroIngredientiNominati è proporzionale al numero di ingredienti quindi lo tratto come m.
La complessità spaziale dipende da numeroColonneNonVuole che è di classe $\Theta(m)$ e salva in ingredientiNominati.
Poi influisce inizializzaClassificaPopolarita che è di classe $\Theta(m)$.
ordinaClassifica è di classe $\Theta(m)$.

Il successivo blocco for è costante nello spazio.
cancellaMatrice è costante nello spazio.
La complessità spaziale dell'algoritmo è $\Theta(m)$.

La complessità temporale di numeroColonneNonVuote è $\Theta(nm)$.
La complessità temporale della stampa è costante.
La complessità temporale di inizializzaClassificaPopolarita è $\Theta(nm)$.
La complessità temporale di ordinaClassifica (numero colonne classifica è m e numero righe 4) è $\Theta(4m + mlogm) = \Theta(mlogm)$ nel caso migliore e $\Theta(4m + m^2logm) = \Theta(m^2logm)$ nel caso peggiore.
Il ciclo for successivo effettua m cicli in cui chiama le funzioni leggiValoreTabella e leggiValoreMatrice più fprinf, tutte funzioni costanti nel tempo, quindi il blocco è $\Theta(m)$.
La complessità temporale di cancellaMatrice è $\Theta(4m) = \Theta(m)$.
La complessità temporale della funzione è $\Theta(nm + nm + mlogm + m + m) = \Theta(nm + mlogm)$ nel caso migliore, $\Theta(nm + nm + m^2logm + m + m) = \Theta(nm + m^2logm)$ nel caso peggiore.

6.8 inizializzaClassificaEsigenza

6.8.1 Variabili e Strutture Dati

Le variabili definite sono *i*, *j* INT iterabili per cicli, *numeroColonnePreferenze* INT conterrà il numero di ingredienti e *preferenzaEspressa* INT legge il contenuto della matrice preferenza mentre viene scorsa.

6.8.2 Algoritmo - Idea

L'idea è scorrere tutta la matrice delle preferenze e salvare nella matrice classifica gli indici dei clienti analizzati, il numero di preferenze forti e il numero di gradimenti espressi (tre righe per ogni cliente).

6.8.3 Algoritmo - Tecnica

Controllo che il numero di clienti sia diverso da 0, in caso contrario lancio errore e termina l'esecuzione.
Inizializzo la matrice classifica per avere 3 righe ed un numero di colonne pari al numero di clienti.
Scorro la prima riga della matrice ed inserisco gli indici da 0 a numeroClienti-1 che faranno da *foreignKey* per la tabella clienti.
Scorro per righe la matrice preferenze, ovvero, effettuo un ciclo per ogni cliente ed in ogni istanza effettuo un ciclo per ogni colonna delle preferenze (numero di ingredienti). Leggo il valore della matrice preferenze nell'entrata i,j, se si tratta di ESCLUSO o RICHIESTO aggiungo 1 alla riga 1 se si tratta di GRADITO aggiungo 1 alla riga 2.

6.8.4 Complessità

La funzione inizializzaMatrice ha una complessità spaziale di $\Theta(3n) = \Theta(n)$ e salva in classifica.
Tutte le altre funzioni chiamata e le variabili definite hanno complessità spaziale costante.
La complessità spaziale della funzione è $\Theta(n)$.

La complessità temporale del primo blocco if è costante.
La funzione inizializzaMatrice ha complessità temporale dell'ordine $\Theta(3n) = \Theta(n)$.
La funzione numeroColonneMatrice è costante.
Il ciclo for successivo effettua n chiamate alla funzione inserimentoElementoMatrice che è costante quindi il ciclo è $\Theta(n)$.
Il ciclo for successivo effettua n chiamate ad un ciclo for che effettua m chiamate.

Quello più interno chiama leggiValoreMatrice che è costante e aggiungiElementoMatrice che è costante, quindi il blocco di due for ha complessità $\Theta(nm)$.
In definitiva la funzione ha complessità $\Theta(n + n + nm) = \Theta(nm)$.

6.9 calcolaOrdineEsigenza

6.9.1 Variabili e Strutture Dati

Le variabili definite sono *classifica* MATRICE utilizzata per ordinare tramite la funzione ordinaClassifica, *numeroClienti* INT salva il numero di clienti (righe di preferenze), *i*, *j* INT iterabili per cicli.

6.9.2 Algoritmo - Idea

L'idea è costruire una matrice 3 x numero di clienti in cui mettere nella prima riga gli indici dei clienti, nella seconda il numero di preferenze forti che esprire e nella terza il numero di gradimenti.
Ordinare la classifica secondo le regole espresse nel PDF del problema.
Satampare il risultato con nome del cliente, numero preferenze forti e numero gredimenti.

6.9.3 Algoritmo - Tecnica

Leggo il numero di righe della matrice preferenze, ovvero il numero di clienti.
Stampo il numero di clienti seguito dalla parola clienti.
Creo una matrice 3xnumeroClienti tramite la funzione inizializzaClassificaEsigenza.
Ordino la classifica tramite la funzione ordinaClassifica.
Stampo il risultato con prima il nome del cliente (preso come la stringa nella tabella clienti con indice l'elemento nella riga 0 della colonna in lettura di classifica), poi il numero di preferenze forti e poi il numero di gradimenti nella stessa riga, questo per ogni cliente.
Infine dealloco la matrice classifica.

6.9.4 Complessità

La complessità spaziale dipende da inizializzaClassificaEsigenza che salva nella variabile classifia 3xnumeroClienti valori quindi ha complessità $\Theta(3n) = \Theta(n)$.
La complessità spaziale di ordinaClassifica è $\Theta(n)$.
Quindi la complessità spaziale della funzione è $\Theta(n)$.

La funzione numeroRigheMatrice ha complessità temporale $\Theta(1)$.
La funzione fprintf è costante.
La funzione inizializzaClassificaEsigenza ha complessità temporale $\Theta(nm)$.
La funzione ordinaClassifica ha complessità temporale (lughezza di classifica è n e altezza 3) $\Theta(3n + nlogn) = \Theta(nlogn)$ nel caso migliore e $\Theta(3n + n^2logn) = \Theta(n^2logn)$ nel caso peggiore.
Il ciclo for successivo fa n chiamate alle funzioni fprintf, leggiValoreTabella e leggiValoreMatrice che sono costanti e 3 chiamate a fprintf e leggiValoreMatrice sempre costanti, il ciclo for ha quindi complessità $\Theta(n)$.
Infine cancellaMatrice ha complessità temporale $\Theta(3n) = \Theta(n)$.
La complessità temporare della funzione è quindi $\Theta(nm + nlogn + n + n) = \Theta(nm + nlogn)$ nel caso migliore e $\Theta(nm + n^2logn + n + n) = \Theta(nm + n^2logn)$ nel caso peggiore.

6.10 inizializzaClassificaIngrEsclusi

6.10.1 Variabili e Strutture Dati

Le variabili definite sono i, j INT iterabili per cicli, *numeroRighePreferenze* INT numero dei clienti e *preferenzaEspressa* INT salva il dato letto dalla matrice preferenze.

6.10.2 Algoritmo - Idea

L'idea è creare una matrice 2xnumeroIngredienti, nella prima riga mettere gli indici degli ingredienti e nella seconda riga mettere il numero di clienti che lo escludono il dato ingrediente.

6.10.3 Algoritmo - Tecnica

Controllo che non siano 0 gli ingredienti, in caso contrario lancio errore e termina l'esecuzione.
Creo la matrice classifica di dimensione 2xnumeroIngredienti.
Salvo in una variabile il numero di righe della matrice preferenze, ovvero il numero di clienti.
Effettuo poi un ciclo for che chiama m volte la funzione aggiungiElementoMatrice così da mettere tutti gli indici da 0 a $m-1$ nella prima riga di classifica.
Per ogni clico chiamo un ciclo for di n istanze dove lweggo il valore della matrice preferenze in i, j e se è escluso incremento il numero di esclusioni dell'ingrediente di indice i.

6.10.4 Complessità

La complessità spaziale di inizializzaMatrice è $\Theta(2m) = \Theta(m)$.
Tutte le altre funzioni sono costanti nello spazio.
Quindi la complessità spaziale della funzione è $\Theta(m)$.

La complessità temporale del primo blocco if è costante.
La complessità temporale di inizializzaMatrice è $\Theta(2m) = \Theta(m)$.
La complessità temporale di numeroRigheMatrice è costante.
Il primo ciclo for effettua m chiamate alla funzione aggiungiElementoMatrice che è costante e al ciclo for di n chiamate alle funzioni leggiValoreMatrice e aggiungiElementoMatrice che sono costanti.
Quindi i due blocchi for annidati hanno complessità temporale $\Theta(nm)$.
La complessità temporale della funzione è quindi $\Theta(m + nm) = \Theta(nm)$.

6.11 ingredientiEsclusiPerNumeroClienti

6.11.1 Variabili e Strutture Dati

Le variabili definite sono *numeroIngredienti* INT numero degli ingredienti, i INT iterabile per ciclo, *counter* INT numero di ingredinenti per dato numero di esclusioni, *numeroInConfronto* INT numero di esclusioni che stiamo considerando, *nuovoNumeroInConfronto* INT indice che si muove nella classifica leggendo i vari nuemeri di esclusione e *classificaIngredientiPerEsclusioni* MATRICE matrice che creo con la funzione precedente per avere il numero di clienti che escludono ogni cliente.

6.11.2 Algoritmo - Idea

L'idea è costruire una matrice 2 x numeroIngredienti con le informazione di esclusione tramite la funzione inizializzaClassificaIngrEsclusi.
Ordinarla per avere vicini i vari ingredienti che sono esclusi dallo stesso numero di clienti.
Scorrere la classifica e contare quanti ingredienti hanno lo stesso numero di clienti che li escludono,

poi stampare questo dato in ordine crescente di numero di clienti.

6.11.3 Algoritmo - Tecnica

Leggo il numero di ingredienti grazie alle colonne della matrice preferenze.
Creo la matrice classifica grazie alla funzione inizializzaClassificaIngrEsclusi.
Ordino la classifica tramite un heapSortCustom.
Stampo la parola esclusioni.
Setto il counter (numero ingredienti che sono esclusi dallo stesso numero di clienti) a 0, setto numeroInConfronto come il primo numero nella riga 1 della classifica (il minimo numero di esclusioni poste da un cliente).
Ciclo per il numero di ingredienti (numero colonne classifica), setto nuovoNumeroInConfronto come il valore nella riga 1 della colonna i, se è cambiato, ovvero se il numero di clienti che escludono un dato ingrediente è cambiato, stampo il numero di ingredienti (counter) ed il numero di esclusioni (numeroInConfronto), risetto il couter a 0. Indipendentemente dal fatto che la condizione del blocco if sia soddisfatta, incremento counter di 1.
Se ho scroso tutti gli elementi ma non ho risettato il counter a 0, vuol dire che devo ancora stampare l'ultimo gruppo e lo faccio nell'ultimo if.
Infine dealloco la matrice classificaIngredientiPerEsclusioni.

6.11.4 Complessità

La complessità spaziale dipende solo da inizializzaClassificaIngrEsclusi, che agisce su classificaIngredientiPerEsclusioni, che ha complessità $\Theta(m)$.

La complessità temporale di numeroColonneMatirce è costante.
La complessità temporale di inizializzaClassificaIngrEsclusi è $\Theta(nm)$.
La complessità temporale di heapSortCustom è $\Theta(2mlogm) = \Theta(mlogm)$.
La complessità temporale di fprintf è costante.
La complessità temporale di leggiValoreMatrice è costante.
Il ciclo for effettua m chiamate alla funzione leggiValoreMatrice che è costante nel tempo e al più m chiamate a fprintf.
Il ciclo for ha quindi complessità temporale di $\Theta(m)$.
La complessità temporale di cancellaMatrice è $\Theta(2m) = \Theta(m)$.
Quindi la complessità temporale della funzione è $\Theta(nm + mlogm + m + m) = \Theta(nm + mlogm)$

6.12 calcoloCoppieIncompatibili

6.12.1 Variabili e Strutture Dati

Le variabili definite sono *numIncompatibili* INT counter numero di coppie incompatibili, *numeroClienti* INT numero dei clienti, i, j e k INT iterabili per cicli, *numeroColonnePreferenze* INT numero di ingredienti, *preferenzaEspressaCli1*, *preferenzaEspressaCli2* INT preferenze (ESCLUSO, RICHIESTO o GRADITO) della coppia in analisi, *coppieIncompatibili* MATRICE matrice che avrà dimensione numeroCoppieIncompatibili x 2 in cui saranno elencate le coppie incompatibili.

In questa funzione uso per la prima volta una matrice simmetrica, perché come spiegato prima, nel confronto tra clienti non avrebbe senso istanziare una matrice intera in quanto l'incompatibilità è una proprietà riflessiva.

6.12.2 Algoritmo - Idea

L'idea è costruire una matrice simmetrica che nell'entrata (i,j) abbia se i due clienti relativi agli indici sono compatibili o meno.
Poi tenere traccia di tutte le coppie incompatibili e stamparle.

6.12.3 Algoritmo - Tecnica

Salvo in una variabile il numero di clienti dal numero di righe della matrice preferenze.
Creo una matrice simmetrica di dimensione numeroClienti.
Salvo in una variabile il numero delle colonne della matrice preferenze, ovvero il numero di ingredienti.
Inizializzo una matrice che conterrà tutte le coppie incompatibili, parto da una riga, 2 colonne sono per gli indici della coppia.
Ciclo per ogni cliente, lo confronto con ogni cliente di indice successivo (non avrebbe senso confrontarlo con indici precedenti, ripeterei il confronto e basta). Per ogni coppia ciclo per ogni ingrediente, salvo le preferenze espresse dai due clienti sullo stesso ingrediente, se sono incompatibili (uno ESCLUSO e l'altro RICHIESTO), metto a 1 l'incompatibilità nella matrice simmetrica.
Se ho già inserito una coppia di incompatibili nella matrice coppieIncompatibili allora aggiungo una riga.
Inserisco nell'ultima riga della matrice coppieIncompatibili gli indici della coppia analizzata ed incremento il numero delle coppie incompatibili.
Blocco il ciclo degli ingredienti in quanto ho già appurato l'esistenza di un ingrediente per cui la coppia di clienti è incompatibile.
Stampo poi il numero delle coppie incompatibili seguite dalle parole coppie incompatibili.
Ciclo per tutta la matrice coppieIncompatibili e stampo i nomi dei clienti utilizzando gli indici nella matrice come *foreignKey* della tabella clienti.
Infine dealloco la memoria di coppieIncompatibili.

6.12.4 Complessità

La complessità spaziale data da inizializzaMatriceSimmetrica che alloca la memoria per le coppie incompatibili, di classe $\Theta(n(n+1)/2)$.
La complessità spaziale data da inizializzaMatrice e aggiungiRigaVuotaMatrice porta al più ad avere $n(n-1)/2-n$ coppie incompatibili (tutti incompatibili con tutti tranne che con sè stessi).
Quindi la complessità spaziale è dell'ordine $O(n(n+1)) = O(n^2)$.

La complessità temporale di numeroRigheMatrice è costante.
La complessità temporale di inizializzaMatriceSimmetrica è $\Theta(n(n+1)/2) = \Theta(n^2)$.
La complessità temporale di numeroColonneMatrice è costante.
La complessità temporale di inizializzaMatrice è costante dato che si tratta di una matrice di base 1x2.
Il primo ciclo ed il secondo effettuano in totale $n(n-1)/2-n$ cicli.
Il ciclo for più interno effettua m cicli.
La funzione leggiValoreMatrice è costante nel tempo.
La funzione inserisciValoreMatriceSimmetrica è costante nel tempo.
La funzione aggiungiRigaVuotaMatrice ha complessità temporale $\Theta(n)$ e nel peggiore dei casi viene chiamata ad ogni coppia di clienti, nel migliore non viene chiamata mai. In generale viene chiamata solo una volta nel ciclo degli ingredienti, il peggior caso vede ultimo l'ingrediente che rende incompatibile la coppia, quindi avvengono comunque m cicli prima di chiamare il break.
La funzione inserimentoElementoMatrice è costante nel tempo.

Questo porta il blocco di 3 cicli for ad avere complessità temporale nel miglior caso $\Theta((n(n-1)/2)m) = \Theta(n^2m)$ e nel peggior caso $\Theta((n(n-1)/2)(m+n)) = \Theta(n^2m+n^3)$.
La funzione fprintf è costante nel tempo.
Effettuo un ciclo di un numero di chiamate pari al numero di coppie incompatibili, al più $n(n-1)/2-n$, chiamando le funzioni fprintf, leggiValoreTabella e leggiValoreMatrice che sono costanti nel tempo.
Quindi la complessità temporale è nel miglior caso costante e nel peggiore $\Theta(n^2)$ Infine la funzione cancellaMatrice ha complessità temporale proporzionale al numero di coppie incompatibili, ovvero al più $n(n-1)/2-n$.
Quindi la complessità temporale è nel miglior caso costante e nel peggiore $\Theta(n^2)$.
La complessità temporale della funzione è quindi, nel miglior caso, $\Theta(n^2+n^2m) = \Theta(n^2m)$ e nel peggior caso $\Theta(n^2+n^2m+n^3+n^2+n^2) = \Theta(n^2m+n^3)$.

6.13 calcoloNumeroIncompatibiliPerCliente

6.13.1 Variabili e Strutture Dati

Le variabili definite sono *i*, *j* INT iterabili per cicli.

6.13.2 Algoritmo - Idea

L'idea è creare una classifica di due righe in cui nella prima siano salvati gli indici di tutti i clienti e nella seconda il numero di clienti con cui esso è incompatibile.

6.13.3 Algoritmo - Tecnica

Effettuo un ciclo per ogni cliente ed inserisco nella riga 0 il suo indice, poi effettuo un ciclo dal cliente 0 al cliente attuale-1, se la coppia è incompatibile allora aggiungo alla seconda riga 1, nella colonna relativa ai due clienti nella coppia (incremento il numero di clienti con cui essi sono incompatibili).

6.13.4 Complessità

La complessità spaziale è $\Theta(1)$.

La complessità temporale è data dai due cicli che effettuano $n(n-1)/2-n$ chiamate a funzioni costanti nel tempo.
La complessità temporale è quindi $\Theta(n^2)$.

6.14 calcoloStimaInfNumeroPizza

6.14.1 Variabili e Strutture Dati

Le variabili definite sono *classificaIncompatibili* MATRICE classifica di due rige in cui la priamo sono gli indici dei clienti e nella seconda il numero di clienti con cui sono incompatibili, *incompatibiliA-Coppie* PIZZA (nome del tipo un po' imporprio) lista dei clienti che generano il numero minimo di pizze, *indiceMovente* INGREDIENTE puntatore ai blocchi della lista che faccio muovere sulla lista per leggerla, *incompatibiliACoppieMatrice* MATRICE matrice (1 dimensionale, quindi vettore) in cui metto i clienti che generano il numero minimo di pizze, *i*, *j* INT iterabili per i cicli, *totaleClienti*, *clientiEffettivi* INT numero di clienti totali e numero di clienti che generano il minimo di pizze, *indiceIncompatibileAccertato* INT indici dei clienti da confrontare per l'incompatibilità, *greedyContinua*

INT indice della condizione di uscita o meno dal greedy, *incompatibileConPrecedenti* INT indice utilizzato per dire se la coppia in analisi è o meno compatibile e *indiceNuovoIncompatibile* INT indice da aggiungere alla lista dei clienti che causano il numero minimo di pizze.

6.14.2 Algoritmo - Idea

L'idea è trovare il numero minimo di pizze necessario e l'elenco dei clienti che causano questo numero.
Per farlo mi creo una classifica dei clienti con il numero di clienti con cui sono incompatibili.
Prendo l'elemento massimo nella classifica, quello con più incompatibili, e lo aggiungo alla lista incompatibiliACoppie.
A questo punto modifico il suo *score* di incompatibilità mettendo -1, numero che sicuramente porterà il cliente aggiunto a non essere il nuovo massimo.
Ora dal secondo giro aggiorno la classifica mandando a -1 lo score dei clienti compatibili con quello aggiunto.
Prendo il massimo rimanente (il cliente con più incompatibili ed incompatibile con quello scelto in precedenza) e lo aggiungo alla lista dei clienti.
Continuo con questo processo fino a che lo score di tutti è -1 (ovvero tutti i clienti sono stati scelti o erano compatibili con almeno un cliente scelto).
Poi trasporto la lista dei clienti in una matrice 1-dimensionale, la ordino con lessico grafico e poi stampo il numero di pizze (numero di clienti nella lista) e l'elenco orindinato di clienti.
Infine dealloco la memoria utilizzata.

6.14.3 Algoritmo - Tecnica

Mi salvo in una variabile il numero di clienti.
Creo una classifica di due righe, una per gli indici dei clienti ed una per il numero di clienti con cui sono incompatibili.
Setto il numero di clienti nella lista pari a 0.
Parte un ciclo while che continuerà finché greedyContinua è uguale a 1.
Setto greedyContinuea a 0.
Se ho almeno un cliente allora piazzo l'indiceMovente all'inizio della lista incompatibiliACoppie, effettuo un ciclo per ogni cliente e setto che di base il clienti i sarà incompatibile con quelli presenti nella lista, ciclo per tutta la lista leggendo l'indice di un cliente in lista, poi muovendo indiceMovente verso il successivo in lista e confrontando quello letto con il clienti i, se sono compatibili cambio incompatibileConPrecedenti a 0 ed esco dal confronto con i clienti in lista.
Alla fine del confronto di i con i clienti in lista, se è risultato compatibile metto il suo score a -1 così che non possa essere aggiunto alla lista, se no metto greedyContinua a 1.
Se sono al primo giro metto di base greedyContinua a 1.
Se greedyContinua è 1 (quindi esiste ancora un cliente da aggiungere alla lista) prendo il cliente con il massimo numero di incompatibilità e lo aggiungo alla lista (se è il primo creo la lista).
Incremento il numero di clienti in lista.
Finito il ciclo trasporto il la lista in una matrice 1 dimensionale (vettore) e la ordino (essendo 1 dimensionale l'ordine sarà direttamente lessicografico).
Infine stampo il numero di pizze e i clienti che erano in lista in ordine alfabetico utilizzando la matrice ordinata ed i nomi nella tabella clienti.
Come ultima cosa dealloco la memoria della lista e delle due matrici usate.

6.14.4 Complessità

La complessità spaziale dipende da inizializzaMatrice $\Theta(2n) = \Theta(n)$.
La complessità spaziale di calcoloNumeroIncompatibiliPerCliente è $\Theta(1)$.
Nel ciclo while le componenti che hanno un contributo sulla complessità spaziale solo creaLista e inserisciNodoFondoLista, costanti nello spazio, che essendo ripetute tante volte quanto clientiEffettivi crescono linearmente con questa variabile, nel caso peggiore essa è pari al numero di clienti n.
Quindi il contributo del ciclo while è $O(n)$.
La funzione listaInMatrice ha complessità spaziale pari a $O(n)$ in quanto è lineare rispetto a clientiEffettivi.
La funzione ordinaClassifica ha complessità spaziale pari a $O(n)$ in quanto è lineare rispetto a clientiEffettivi.
La complessità spaziale della funzione è quindi $O(n + n + n + n) = O(n)$.

La complessità temporale di leggiGrandezzaMatriceSimmetrica è costante.
La complessità temporale di inizializzaMatrice è $\Theta(2n) = \Theta(n)$.
La complessità temporale di calcoloNumeroIncompatibiliPerCliente è $\Theta(n^2)$.
Per il ciclo while il caso migliore è quello in cui sono tutti compatibili così il for interno effettua una sola operazione per ciclo, il caso peggiore è quello in cui sono tutti incompatibili quindi vengono effettuati tutti i confronti per ogni ciclo.
Nel caso migliore il primo for effettua n cicli, il secondo for ne effettua 1 sono, e la funzione inserimentoElementoMatrice è costante nel tempo.
Il blocco if finale viene chiamato 1 volta sola, il primo giro, e indiceMassimoRispettoARigaMatrice è $O(n)$, creaLista è costante e inserisciNodoFondoLista è costante.
Tutto questo viene fatto 1 volta perché greedyContinua = 0 dopo il ciclo for.
Quindi il caso migliore ha complessità temporale $O(n + n) = O(n)$.
Analizzo allora il caso peggiore.
Il primo ciclo for avvine n volte mentre il ciclo for interno ogni volta ha una iterazione aggiuntiva partendo da 1 fino a n, le funzioni che chiama al suo interno sono costanti nel tempo.
La funzione inserimentoElementoMatrice è costante nel tempo.
Quindi il ciclo for ha complessità $\Theta(1 + 2 + ... + n) = \Theta(n(n + 1)/2) = \Theta(n^2)$.
Il blocco if finale viene chiamato ad ogni ciclo while, la sua complessità temporale è $O(n)$.
Il ciclo while verrà effettuato n volte.
Quindi il caso peggiore ha complessità temporale $O(n^3 + n^2) = O(n^3)$.
La complessità temporale di listaInMatrice è $O(n)$ essendo lineare con clientiEffettivi.
La complessità temporale di ordinaClassifica è $O(n^2 logn)$ essendo lineare con clientiEffettivi e prendendo il caso peggiore.
La complessità delle stampe successive è costante.
La complessità temporale di cancellaLista è $O(n)$ essendo lineare con clientiEffettivi.
La complessità temporale di cancellaMatrice per incompatibiliACoppiMatrice è $O(n)$ essendo lineare con clientiEffettivi.
La complessità temporale di cancellaMatrice per classificaIncompatibili è $\Theta(2n) = \Theta(n)$.
Quindi la complessità della funzione è $O(n + n^2 + n + n + n^2 logn + n + n) = O(n^2 logn)$ nel caso migliore e $O(n + n^2 + n^3 + n + n^2 logn + n + n) = O(n^3)$ nel caso peggiore.

6.15 aggiornaListaClientiSoddisfatti

6.15.1 Variabili e Strutture Dati

Le variabili definite sono i INT iterabile per ciclo, *numeroClienti* INT numero dei clienti, *numeroIngredienti* INT numero degli ingredienti e indiceClienteInsoddisfatto INGREDIENTE (tipo linevemente

imporoprio) salva il puntatore ad un cliente da togliere dalla lista dei clienti soddisfatti.

6.15.2 Algoritmo - Idea

L'idea è, dato l'indice di un ingrediente, togliere da una lista di clienti tutti quelli che lo escludono, decrementando anche il numero dei clienti soddisfatti.

6.15.3 Algoritmo - Tecnica

Salvo in una variabile il numero delgi ingredienti.
Controllo che l'indice dell'ingrediente da confrontare sia maggiore di 0 e mino di numeroIngredienti, in caso contrario lancio errore e termina l'esecuzione.
Salvo il numero dei clienti in una variabile.
Ciclo per ogni cliente e se la sua entrata nella tabella preferenze, rispetto all'ingrediente fissato, è ESCLUSO, allora, se il numero di clienti soddisfatti è (diventato) 0, ritorno la lista dei clienti soddisfatti, se no trovo l'indice del cliente nella lista e lo tolgo dalla lista, con l'accortezza che se si tratta del primo della lista, il puntatore clientiSoddisfatti dovrà puntare al secondo della lista.
Decremento il numero di clienti soddisfatti.
Finito il ciclo, se non sono uscito prima (quindi se la lista di clienti soddisfatti è maggiore di 0), ritorno la lista.

6.15.4 Complessità

La complessità spaziale è costante in quanto nessuna funzione chiamata ha complessità magiore e le variabili nella funzione hanno dimensione prefissata.
Quindi la complessità spaziale è $\Theta(1)$.

La complessità temporale di numeroColonneMatrice è costante.
La complessità temporale del blocco if è costante.
La complessità temporale di numeroRigheMatrice è costante.
Viene poi effettuato un ciclo for di n istanze, nel caso migliore, dove nessun cliente esclude l'ingrediente, non effettua operazioni, quindi ha complessità $\Theta(n)$.
Nel caso peggiore, dove tutti i clienti escludono l'ingrediente, la funzione indiceDatoContenutoInLista viene chiamata ogni volta ed ha complessità temporale $\Theta(n)$ (in realtà ogni volta decresce di 1, fino ad arrivare ad 1).
Quindi nel caso peggiore ha complessità $\Theta(n + (n-1) + ... + 1) = \Theta(n^2)$.
La funzione quindi ha complessità temporale $\Theta(n)$ nel miglior caso e $\Theta(n^2)$ nel peggior caso.

6.16 modificaPreferenzeIngrediente

6.16.1 Variabili e Strutture Dati

Le variabili definite sono *i*, *j* INT iterabili per cicli, *numeroClienti* INT numero dei clienti e *numeroIngredienti* INT numero degli ingredienti

6.16.2 Algoritmo - Idea

L'idea è scorrere i vari clienti e ogni qual volta trovo un cliente che esclude l'ingrediente passato come parametro modificare tutte le due proferenze (su ogni ingrediente) mettendo NESSUNA_RELAZIONE.
In più segno che ogni cliente esclude l'ingrediente passato come paramentro.
La prima parte serve per far sì che i clienti che escludevano un ingrediente aggiunto alla pizza non vengano considerati nel ricalcolo della quantità di clienti che escludono un determinato ingrediente

(non pesano più sulla scelta degli ingredienti).
La seconda parte serve per non permettere di riconsiderare l'ingrediente già scelto.

6.16.3 Algoritmo - Tecnica

Salvo il numero di ingredienti in una variabile, numero colonne preferenze.
Se l'indice dell'ingrediente passato come parametro è minore di 0 o maggiore o uguale al numero di ingredienti lancio errore e termina l'esecuzione.
Salvo il numero di clienti in una variabile, numero righe preferenze.
Ciclo per ogni cliente, controllo se esso esclude l'ingrediente passato come parametro, se lo fa ciclo per ogni ingrediente e metto che il cliente i non ha nessuna preferenza su nessun ingrediente.
Indipendentemente dalla condizione dell'if metto che il cliente i esclude l'ingrediente paramentro (questo porterà ogni cliente ad escludere l'ingrediente).

6.16.4 Complessità

La complessità spaziale è $\Theta(1)$.

La complessità temporale di numeroColonneMatrice, del blocco if e di numeroRigheMatrice è costante.
Il ciclo for effettua n cicli, nel caso peggiore in cui l'ingrediente è escluso da tutti, viene effettuato ogni volta un ciclo for di m istanze. Nel caso migliore, dove nessuno esclude l'ingrediente, questo ciclo for non viene chiamato mai.
Quindi la complessità temporale nel caso migliore è $\Theta(n)$, mentre nel caso peggiore è $\Theta(nm)$.
La complessità temporale della funzione è nel caso migliore $\Theta(n)$, mentre nel caso peggiore è $\Theta(nm)$.

6.17 creazionePizza

6.17.1 Variabili e Strutture Dati

Le variabili definite sono *minClassificaIngredienti* INT salva l'indice dell'ingrediente meno escluso, *numeroClienti* INT numero dei clienti, *numeroClientiSoddisfatti* INT numero dei clienti soddisfatti per la pizza in costruzione, *numeroClientiSoddisfattiTemp* INT variabile temporanea per tornare indietro di un passo quando il numero dei clienti soddisfatti va a 0, *numeroIngredienti* INT numero degli ingredienti, *numeroIngredientiPizza* INT counter degli ingredienti nella pizza in costruzione, *classificaIngredientiPerEsclusioni* MATRICE classifica in cui ho tutti gli ingredienti ed il numero di clienti che li escludono, *clientiSoddisfattiTemp* PIZZA pizza copia per tornare indietro di un passo quando il numero dei clienti soddisfatti va a 0

6.17.2 Algoritmo - Idea

L'idea è quello di partire da una matrice che lega l'indice di un ingrediente al numero di clienti che lo escludono, ciclare fino a che l'ingrediente escluso meno è escluso da un numero di clienti inferiore al numero totale di clienti (o fino a che non ho usato tutti gli ingredienti), effettuando le seguenti operazioni:
Creare un back-up della lista dei clienti soddisfatti in caso l'aggiunta di un ingrediente alla pizza porti questa lista ad essere vuota.
Aggiornare la lista dei clienti soddisfatti mettendo il vincolo di aggiungere l'ingrediente meno escluso.
In caso il numero di clienti soddisfatti vada a 0, fare un passo indietro grazie al back-up e chiudo il ciclo.
In caso ci siano ancora dei clienti soddisfatti, aggiungo l'ingrediente meno escluso alla lista della pizza.

Risetto le preferenze per far sì che non venga scelto di nuovo lo stesso ingrediente e tale che i clienti esclusi dall'ingrediente già scelto non pesino per le prossime scelte di ingredienti.
Visto che in questo passaggio faccio in modo che l'ingrediente scelto venga escluso da tutti, convergerò ad un momento in cui tutti escludono tutto e quindi il ciclo si ferma.
Ricalcolo la classifica di esclusione degli ingredienti.
Ne trovo l'indice minimo.

6.17.3 Algoritmo - Tecnica

Prendo il numero degli ingredienti dalle colonne della matrice preferenze.
Setto il numero degli ingredienti nella pizza a 0.
Prendo il numero dei clienti dalle righe della matrice preferenze.
Inizializzo la classifica degli ingredienti per esclusioni tramite inizializzaClassificaIngrEsclusi.
Ne trovo il meno escluso e mi salvo l'indice.
Setto il numero dei clienti soddisfatti al numero dei clienti (una pizza vuota soddisfa tutti, in teoria).
Faccio partire un ciclo che terminerà quando il meno escluso sarà escluso da tutti o quando avrò messo tutti gli ingredienti possibili.
Per ogni ciclo:
Effettuo un back-up della lista dei clienti soddisfatti dall'attuale pizza, copiando il numero dei clienti soddisfatti, creando una lista vuota e poi copiando la lista degli attuali clienti soddisfatti.
Aggiorno la lista dei clienti soddisfatti fissanfo l'aggiunta dell'ingrediente meno escluso.
Questo passaggio decrementa anche la variabile numeroClientiSoddisfatti.
Se numeroClientiSoddisfatti è a 0 dopo essere stata decrementata allora ricopio il vecchio numero di clienti soddisfatti e reinserisco la copia della vecchia lista dei clienti soddisfatti nella variabile clienti-Soddisfatti ed interrompo il ciclo avendo trovato la lista minima.
Aggiungo allora l'ingrediente meno escluso alla lista della pizza (con l'accortezza che se si tratta del primo inserisco l'indice dell'ingrediente nel primo nodo già creato).
Incremento il numero di ingredienti.
Ricalcolo le preferenze tramite la funzione modificaPreferenzeIngrediente che, come già detto, permette di non far pesare più i clienti già esclusi e far escludere da tutti l'ingrediente scelto.
Dealloco la classifica degli ingredienti.
Ricreo la classifica degli ingredienti per esclusione.
Dealloco la lista di back-up.
Ritorno l'indirizzo al primo blocco della listta degli ingredienti.

6.17.4 Complessità

La funzione inizializzaClassificaIngrEsclusi ha complessità spaziale $\Theta(m)$, anche se viene chiamata ad ogni ciclo, viene anche deallocata ogni volta.
La funzione creaCopiaLista ha complessità spaziale lineare rispetto al numero di clienti quindi $O(n)$, anche se viene chiamata ad ogni ciclo, viene anche deallocata ogni volta.
Le altre funzioni non pesano sulla complessità spaziale, quindi la complessità spaziale della funzione è $O(n + m)$.

La complessità temporale della funzione numeroColonneMatrice è costante.
La complessità temporale della funzione numeroRigheMatrice è costante.
La complessità temporale della funzione inizializzaClassificaIngrEsclusi è $\Theta(nm)$.
La complessità temporale della funzione indiceMinimoRispettoARigaMatrice è $O(m)$.

Per quanto riguarda il ciclo while:
Il caso migliore è quando tutti gli ingredienti (meno 1 se no è triviale) sono esclusi da tutti i clienti e quello rimanente è escluso da n-1 clienti, questo porta il ciclo while ad avvenire una volta sola.
La funzione creaLista ha complessità temporale costante.
La funzione creaCopiaLista ha complessità temporale lineare con il numero di clienti soddisfatti, in questo caso rimane 1, quindi è costante.
La funzione aggiornaListaClientiSoddisfatti ha complessità $\Theta(n^2)$.
Il blocco if non viene chiamato in questo caso.
La parte di inserimento di inserisciElementoLista o di inserisciNodoFondoLista è costante.
La funzione modificaPreferenzeIngrediente (paradossalmente in questo caso è nel suo caso peggiore) ha complessità $\Theta(nm)$.
La funzione cancellaMatrice ha complessità $\Theta(m)$.
La funzione inizializzaClassificaIngrEsclusi ha complessità $\Theta(nm)$.
La funzione indiceMinimoRispettoARigaMatrice ha complessità $O(m)$.
La funzione cancellaLista ha complessità lineare con il numero di clienti soddisfatti, in questo caso costante.
Il caso peggiore è quando tutti gli ingredienti hanno 0 esclusioni tranne 1 che è escluso da tutti (così quando viene aggiunto entro nel blocco if), quindi il ciclo avviene m volte.
Le complessità temporali rimangono invariate tranne che per creaCopiaLista è $\Theta(n)$, aggiornaListaClientiSoddisfatti ha complessità $\Theta(n)$ tranne che nell'ultimo ciclo in cui ha $\Theta(n^2)$, quindi lo considero come $\Theta(n^2)$, modificaPreferenzeIngrediente ha complessità $\Theta(n)$ e cancellaLista ha complessità $\Theta(n)$. In più viene chiamato il blocco if una volta sola, il quale ha complessità $\Theta(n)$. Quindi la funzione nel caso migliore ha complessità temporale $O(nm+m+n^2+nm+m+nm+m) = O(n^2+nm)$, nel caso peggiore $O(nm+m+m(n^2+n+m+nm+m+n)+n) = O(n^2m+nm^2)$

Un caso utile per dopo è quando tutti gli ingredienti hanno 0 esclusioni, in quel caso la complessità temporale di aggiornaListaClientiSoddisfatti è sempre $\Theta(n)$ e quindi la complessità della funzione diventa $O(nm+m+m(n+n+m+nm+m+n)+n) = O(nm^2)$

6.18 calcoloStimaMenuPizza

6.18.1 Variabili e Strutture Dati

Le variabili definite sono *i*, *j* INT iterabili per i cicli, numeroClienti INT numero dei clienti, *numeroClientiSoddisfattiCiclo* INT numero dei clienti che vengono soddisfatti dalla pizza in costruita, *numeroClientiSoddisfattiTotale* INT counter dei clienti soddisfatti, *numeroIngredienti* INT numero degli ingredientim, *preferenzeCopia* MATRICE copia della matrice preferenze in quanto per creare la pizza viene modificata la matrice che passo come parametro, *numeroPizze* INT counter del numero di pizze, *menuPizze* MENU array di pizze, *clientiSoddisfattiPerPizza* MENU array di liste di clienti soddisfatti, *lettoreClientiDaTogliere* INGREDIENTE indice movente nella lista dei clienti soddisfatti, *pizza*, *clientiSoddisfatti* MATRICE matrici in cui copio gli indici degli ingredienti e dei clienti per ordinarle e poi stamparle.

6.18.2 Algoritmo - Idea

L'idea è greare due array di liste, uno il menu ed uno la lista di clienti soddisfatti per pizza.
Ciclare finchè non ho soddifatto tutti i clienti.
Creare una nuova lista di clienti e partire con tutti i clienti soddisfatti.
Fare un back-up della matrice preferenze.
Creare una nuova lista di ingredienti.
Creare una pizza e posizionare il puntatore al primo blocco nell'array del menu.

Recuperare il numero di clienti soddisfatti con la nuova pizza.
Togliere dai clienti da soddisfare con le nuove pizze quelli soddisfatti con questa.
Incrementare il numero di pizze fatte.
Finito il ciclo stampo il numero di pizze fatte.
Per ognuna ordino in ordine lessicografico gli ingredienti e li stampo e poi ordino in ordine lessicografico i clienti soddisfatti e li stampo.

6.18.3 Algoritmo - Tecnica

Mi salvo il numero di clienti e di infgedienti.
Setto a 0 il numero di clienti soddisfatti totale.
Setto a 0 il numero di pizze fatte.
Inizializzo l'array di pizze (liste) e di clienti soddisfatti (liste).
Inizializzo la matrice di preferenzeCopia che mi farà da backup per ogni pizza, in quanto la creazione della pizza modifica la matrice preferenze che gli viene passata.
Entro in un ciclo che finità quando tutti i clienti saranno soddisfatti.
Parto dal presupposto che tutti i clienti verranno soddisfatti.
Istanzio lo spazio per una nuova lista di clienti soddisfatti e riempo la lista con tutti gli indici dei clienti.
Creo un backup della matrice preferenza.
Istanzio lo spazio per una nuova pizza (lista di ingredienti).
Creo una pizza con creazionePizza e ne salvo l'indirizzo del primo ingrediente (testa della lista) nell'array del menu.
Mi salvo il numero di clienti soddisfatti dalla pizza appena creata.
Per fare in modo che i clienti soddisfatti non possano essere riconsiderati per le prossime pizze, ciclo per ogn'uno di loro e metto in preferenze che che è escludono tutti gli ingredienti.
Infine incremento il numero di pizze.
Fuori dal ciclo stampo il numero di pizze creato.
Per ogni pizza faccio una copia degli ingredienti in matrici e le riordino, poi stampo gli ingredienti.
Faccio la stessa cosa per i clienti soddisfatti.
Dalloco queste matrici ad ogni ciclo.
Infine dealloco lo spazio del back up delle preferenze e degli array di liste.

6.18.4 Complessità

La complessità spaziale di creaMenuLista è $\Theta(n)$.
La complessità spaziale di inizializzaMatrice è $\Theta(nm)$.
Nel ciclo while creazionePizza alloca e dealloca lo spazio quindi la conto una volta sola, $O(n+m)$.
Nella fase di stampa vengono allocate e deallocate ogni volta due matrici, quindi le conto una volta sola, la prima $O(m)$ la seconda è $O(n)$.
Quindi la complessità spaziale della funzione è $O(n+nm+n+m+m+n) = O(nm)$.

La complessità temporale di numeroRigheMatrice è costante.
La complessità temporale di numeroColonneMatrice è costante.
La complessità temporale di creaMenuLista è costante.
La complessità temporale di inizializzaMatrice è $\Theta(nm)$.
Per il ciclo while:
Nel caso migliore, una pizza soddisfa tutti i clienti, ho che il ciclo viene fatto una volta sola.
La complessità temporale di nuovaPizzaInMenu è costante.
Il ciclo for ha n istanze in cui chiama leggiPizzaDaMenu e inserisciNodoFondoLista che sono costanti,

quindi il ciclo è $\Theta(n)$.
La complessità temporale di creaCopiaMatrice è $\Theta(nm)$.
La complessità temporale di nuovaPizzaInMenu è costante.
La complessità temporale di creazionePizza è $O(nm^2)$ in questo caso.
La complessità temporale di cambiaPizza è costante.
La complessità temporale di leggiPizzaDaMenu è costante.
La complessità temporale di recuperaLunghezzaLista è $O(n)$.
La complessità temporale di leggiPizzaDaMenu è costante.
Il ciclo for in questo caso compie n chiamate a:
un ciclo for di m chiamate a leggiElementoLista e inserimentoElementoMatrice funzioni costanti nel tempo
nextIngredienteLista funzione costante nel tempo.
Quindi con complessità $\Theta(nm)$.
Portando la complessità del ciclo while nel caso migliore a $O(n + nm + nm^2 + n + nm) = O(nm^2)$ Nel caso peggiore viene effttuato un ciclo per ogni cliente.
L'unica complessità che cambia è creazionePizza che diventa $O(n^2m + nm^2)$.
Portando la complessità del ciclo while nel caso peggiore a $O(n(n + nm + n^2m + nm^2 + n + nm)) = O(n^3m + n^2m^2)$.
Il ciclo for di stampa viene effettuato al più n volte se c'è una sola pizza per cliente.
La complessità temporale di recuperaLunghezzaLista è $O(m)$.
La complessità temporale di listaInMatrice è $O(m)$.
La complessità temporale di ordinaClassifica è $O(mlogm)$.
Il ciclo for fa al più m cicli stampando, quindi con funzione costante, ergo $O(m)$.
La complessità temporale di recuperaLunghezzaLista è $O(n)$.
La complessità temporale di listaInMatrice è $O(n)$.
La complessità temporale di ordinaClassifica è $O(nlogn)$.
Il ciclo for fa al più n cicli stampando, quindi con funzione costante, ergo $O(n)$.
La complessità temporale di cancellaLista è $O(m)$.
La complessità temporale di cancellaMatrice è $O(m)$.
La complessità temporale di cancellaLista è $O(n)$.
La complessità temporale di cancellaMatrice è $O(n)$.
Fuori dal ciclo cancellaMatrice è $O(nm)$ ed i due cancellaMenu sono costanti.
Questo porta la funzione ad avere complessità temporale nel caso migliore $O(nm + nm^2 + n(m + m + mlogm + m + n + n + nlogn + n + m + m + n + n) + nm) = O(nm^2 + n^2logn)$ e nel caso peggiore $O(n^3m + n^2m^2 + n(m + m + mlogm + m + n + n + nlogn + n + m + m + n + n) + nm) = O(n^3m + n^2m^2)$.