Sintesi del problema.

Il problema si articola in tre differenti punti. Dato un insieme di N comuni, è da determinare in quali di essi posizionare delle stazioni di ricarica.

Si tratta quindi di trovare delle soluzioni costituite da sottoinsiemi di comuni che verificano particolari condizioni o che siano ottime in relazione a specifiche funzioni di obiettivo.

Strategia algoritmica.

Secondo punto

Per poter determinare il sottoinsieme ottimo di comuni, si ricorre ad un algoritmo ricorsivo che determini l'insieme delle parti dell'insieme di partenza. Per implementarlo si è scelto il modello che calcola il powerset come unione dell'insieme vuoto e di tutti i sottoinsiemi di cardinalità 1,…,N (N: cardinalità dell'insieme di partenza), calcolando questi ultimi come combinazioni semplici di N elementi presi a gruppi di k, con k=1,…,N.

In questo modo, iterando l'algoritmo ricorsivo che determina le combinazioni semplici *con k crescente*, non appena è trovata una soluzione accettabile (si veda sotto per il test sull'accettabilità), quest'ultima è sicuramente ottima, essendo, tra le soluzioni accettabili, quella a cardinalità minore possibile. Seguendo tale approccio (possibile solo usando tale modello per modellare il powerset), il ciclo iterativo è fermato non appena una tale soluzione è trovata, senza continuare l'esplorazione dello spazio degli stati.

Anche l'algoritmo delle combinazioni semplici è fermato non appena è trovata una soluzione accettabile, poiché ulteriori soluzioni non potranno essere migliori, avendo la stessa cardinalità. Per far questo è usata una variabile *stop*, passata by reference alla funzione ricorsiva.

Per quanto riguarda l'accettabilità di una soluzione, si ricorre ad un'apposita funzione di verifica introdotta nel punto primo.

Essa, iterando su ogni comune, controlla la distanza da tutti comuni appartenenti alla soluzione: se almeno una è minore di distMax, il comune si troverà sicuramente a una distanza minore di distMax dalla stazione più vicina. In questo modo non è necessario determinare esplicitamente la stazione più vicina e confrontarne la distanza dal comune. In caso contrario, la soluzione non è accettabile si verifica un'uscita anticipata dal ciclo iterativo.

Terzo punto

Seguendo lo stesso approccio del punto precedente, son determinati i vari sottoinsiemi come combinazioni semplici. Tuttavia, in questo caso, non è sufficiente assumere di poter posizionare una sola stazione in ogni comune appartenente alla soluzione. Dunque, per ogni possibile soluzione, è necessario usare un algoritmo ricorsivo che, seguendo il modello del principio di moltiplicazione, assegni ad ogni comune un numero x di stazioni. Considerati i k comuni appartenenti al sottoinsieme che costituisce la possibile soluzione, per ogni insieme sono possibili m scelte, con m=1,…,M (M: numero massimo di stazioni di ricarica localizzabili).

Seguendo tale approccio, è inoltre possibile, permettendo nelle scelte anche m=0, usare solo il secondo algoritmo, in quanto i possibili sottoinsiemi dell'insieme di partenza sono generati automaticamente, considerando assente dalla soluzione un comune il cui numero di stazioni assegnate è 0.

Il **pruning** consiste nel filtrare le soluzioni parziali: se in esse la somma delle stazioni localizzate nei comuni è già maggiore del numero fisso assegnato `numStaz`, allora quell'albero della discesa ricorsiva è "potato" poiché non potrà portare a soluzioni accettabili.

Sulle possibili soluzioni è inoltre applicata prima una verifica di accettabilità (per verificare che la somma delle stazioni localizzate nei comuni sia pari a `numStaz`), poi una funzione obiettivo che assegna ad ogni possibile soluzione un valore reale q, necessario per poter determinare la soluzione ottima in relazione a tale valore.

Per quanto riguarda quest'ultima funzione obiettivo, essa utilizza il vettore contenente l'elenco dei comuni in cui sono presenti le stazioni (`sol`, nella specifica funzione) e il vettore contenente il numero di stazioni assegnate ad ognuno di essi (`staz`, nella specifica funzione) per calcolare il valore `q` come somma della popolazione di ogni comune, moltiplicata per la distanza dal comune con stazioni più vicino e divisa per il suo numero di stazioni.

Primo punto

Letta da file una soluzione proposta, comprendente l'elenco dei comuni, essendo essa nello stesso formato delle possibili soluzioni trovate al secondo punto, la si testa usando la stessa funzione di verifica ivi introdotta.

Descrizione della struttura dati.

Essendo i comuni identificati da interi progressivi nell'intervallo [0, N-1] non è necessaria un'apposita struttura dati per memorizzarli.
È necessaria invece una struttura dati per memorizzare la soluzione, comprendente l'insieme dei comuni in cui localizzare le stazioni e, nel caso in cui le stazioni assegnate ad un comune possano essere maggiori di uno, un vettore parallelo per memorizzare il numero di stazioni assegnate al comune. L'i-esimo elemento del secondo vettore, determina quante stazioni son state assegnate a sol[i].
Nel caso in cui la soluzione sia (provvisoriamente) ottima, è usata una struttura dati accessoria `bestSol` per memorizzarla rendendone più agevole la memorizzazione e l'accesso.
Entrambi i vettori sono racchiusi in una matrice di 2 righe, dove nel primo elemento di ogni riga è memorizzata la cardinalità del vettore e dal secondo è ricopiato uno dei due vettori precedentemente descritti.

Differenze tra compito cartaceo e codice allegato alla relazione.

- Aggiunta prototipi, *include*, *define*, inserimento delle funzioni in un main, implementazioni funzioni di basso livello (copia e stampa di vettori)
- Modifica dei seguenti delle seguenti righe, dovute a sviste durante il compito in aula
    - Nel ciclo *for* della funzione pm, la condizione di test è: `i<stazCom[val[pos]],` invece di `i<numStaz[val[pos]],` essendo la prima la denominazione corretta (in linea con quanto riportato sulla traccia) del vettore che si voleva considerare
    - All'interno della funzione ob2, a `num` si assegna `num = staz[j]` invece di `num = staz[sol[j]]`
    - All'interno delle funzioni acc1 e acc2, c è incrementato con `c+=sol[i]` invece di `c+=sol[val[i]]`
    - Nel ciclo *for* della funzione pm, la condizione di test presenta un `<=` invece del solo `<` poiché, in accordo con quanto riportato nella traccia, nel vettore sono memorizzati i valori massimi, e non i valori massimi+1