Documentazione Zaino

Prefazione

Questa documentazione descrive la classe zaino e l’implementazione dei suoi metodi ed il main.

In questo documento parleremo di una classe chiamata zaino, aventi come struttura i seguenti tipi:

* vector<int> v, che contengono i coefficienti dei vincoli del problema;
* vector<int> funzione\_obbiettivo, che contiene i coefficienti della funzione obbiettivo;
* vector<int> Vs\_vet, che contiene Le Vs calcolate nel Branch And Bound;
* vector<bool>includi\_vet, che contiene informazioni necessarie per includere ed escludere forzatamente determinate celle di v e funzione\_obbiettivo nell’esecuzione dell’algoritmo Branch And Bound;
* capienza\_massima, un intero che conterrà il peso massimo che lo zaino può sopportare

All’interno della classe ci sono le seguenti funzioni di utilità:

* vector<double> trova\_rendimento(), che mi restituisce un vector contenente i valori dei rendimenti posti in ordine decrescente;
* int Vs\_zaino(int &, vector<int>&, int), una funziona che calcola la Vs, rendendo conto di eventuali ammissioni esclusioni forzate di indici nel Branch And Bound. La funzione aggiorna il vettore delle ammissioni di appoggio e restituisce l’indice del valore decimale che sarà da includere o escludere successivamente;
* static void togli\_1(vector<int>&), che resetta il vettore delle ammissioni settati ad 1;
* void includi(vector<int>&,int), che include una cella di v e funzione\_obbiettivo, settando 2 al vettore delle ammissioni;
* void escludi(vector<int>&,int), che esclude una cella di v e funzione\_obbiettivo, settando -1 al vettore delle ammissioni;
* void costruisci\_albero(vector<int>&,int,int,int \*,int = 0), che calcola le Vs di ogni nodo del Branch And Bound;
* void aggiorna\_vettore\_Vs(int), che aggiorna le Vi correnti, secondo la logica della convenzione spiegata a lezione, tagliando eventuali rami del branch and Bound;
* void taglia(bool\*&,int), che setta lo stato di tagliato al nodo e ai suoi figli correnti;
* int trova\_Vi(int\*,int,int,bool \*), che restituisce un valore di Vi a seconda dell’altezza dell’albero e a seconda del valore delle Vs calcolate in precedenza;

Ci sono le seguenti funzioni pubbliche:

* zaino(), costruttore;
* ~zaino(),distruttore;
* void risolvi\_zaino\_intero(), che calcola la Vs una sola volta secondo il modello del problema dello zaino intero;
* void risolvi\_zaino\_binario(int &, int &, int &), che calcola sia la Vi, che la Vs del problema dello zaino binario;
* void Branch\_Bound(), che esegue il Branch And Bound, secondo il modello del problema dello zaino binario;

main.cpp

In questo file si chiama il costruttore zaino() e poi l’esecuzione finisce.

Alcuni Metodi Importanti

Zaino()

Il costruttore zaino inizializza includi\_vet con celle di valore nullo e Vs\_vet con celle di valore casuale, entrambe di dimensione 15, con l’assunto che l’albero non abbia più 4 piani(inclusa la radice).

Si chiede all’utente di indicare la dimensione dei vettori v e funzione\_obbiettivo con un solo input, per poi indicare nelle righe successive, in questo ordine:

* di scrivere in ordine i valori della funzione obbiettivo;
* di scrivere in ordine i vincoli(i pesi) della funzione obbiettivo, che avrà la dimensione che era già stata decisa dall’utente;
* di scrivere la capienza massima del problema, che avrà la dimensione che era già stata decisa dall’utente.

Successivamente appariranno, stampati a schermo, le seguenti opzioni:

1. zaino intero, soltanto;
2. Branch And Bound con zaino binario all'inizio;
3. solo zaino binario;
4. zaino intero + Branch And Bound con zaino binario all'inizio.

Digitando in input il numero corrispondente all’elenco verranno eseguite le istruzioni come indicato nell’elenco, precisamente, secondo questa regola:

* per calcolare lo zaino intero verrà eseguita la funzione risolvi\_zaino\_intero, che stamperà la soluzione del problema;
* per calcolare lo zaino binario verrà eseguita la funzione risolvi\_zaino\_binario, che stamperà la soluzione del problema;
* per eseguire il branch and bound, verrà chiamata la funzione Branch\_Bound, che stamperà la soluzione del Branch And Bound.

In particolare, ci teniamo a precisare che il formato della soluzione sarà il seguente:

* verrà indicato il modello del problema e verranno indicati per la valutazione inferiore e superiore una soluzione composta dal vettore soluzione, che sarà 0, se non incluso, 1 altrimenti e una frazione, un valore PARZIALMENTE incluso, con poi indicanti i valori della Valutazione Inferiore(indicata con Vi) e della Valutazione Superiore(indicata con Vs), se si chiamano le corrispondenti funzioni pubbliche
* verrà stampato il formato del Branch e Bound con il seguente formato:

1. Per la radice c’è il formato P:[Vi,Vs]
2. Per gli altri c’è il formato Pij:[Vi,Vs], dove i è l’altezza dell’albero e j la disposizione da sinistra verso destra dei nodi, fissata l’altezza i-esima
3. Se il nodo ha un taglio, verrà specificato che sarà tagliato e per quali ragioni esso accada.

Entriamo in merito alla funzione Branch\_Bound

Void Branch\_Bound()

Questa funzione è la parte più grossa del progetto, in quanto usa grandissima parte delle funzioni di utilità presenti nella classe.

In tanto si chiama la funziona pubblica risolvi\_zaino\_binario, così da risolvere il problema dello zaino binario senza inclusioni ed esclusioni forzate, con il formato della stampa già descritta nella sezione precedente.

Dopo di che si istanzia un vector locale chiamato ammesso che servirà a far ricordare all’algoritmo quali indici includere ed escludere, inizializzato a 0 e di dimensione pari al vector v della classe.

Ogni cella di ammesso può avere i seguenti valori:

1. -1, che indica che l’indice è già escluso forzatamente prima dell’applicazione dell’algoritmo dello zaino binario nel passo corrispondente del Branch And Bound;
2. 0, che indica che l’indice non è ancora stato incluso nell’algoritmo dello zaino binario;
3. 1, che indica che l’indice è stato incluso durante l’algoritmo dello zaino binario, con il fine di calcolare la Vs
4. 2, che indica che l’indice è stato forzatamente incluso prima dell’applicazione dell’algoritmo dello zaino binario

Grazie a questo riusciamo a calcolare ad ogni singola iterazione la Vs, senza commettere errori.

Poi si chiama una funzione chiamata costruisci\_albero, che, preso come argomenti il vettore ammesso, una posizione pos di partenza pari a 0, e una variabile intera di appoggio e un puntatore di appoggio, ricostruisce il risultato del Branch and Bound, calcolando tutte le Vs, fino al 4° piano, salvandoli in Vs\_vet della classe. I risultati saranno sufficienti per trovare casi in cui la Vs sia diversa dalla Vi, senza alcuni aggiornamenti: in tal caso il valore corrente all’interno della sua corrispettiva cella è negativa.  
Alla fine della chiamata della funzione appena descritta, verrà invocato il metodo aggiorna\_vettore\_Vs, che preso in input la prima Vi calcolata, si occupa di aggiornare la Vi e tagliare i rami del branch and bound, secondo questi criteri:

1. Si chiamano localmente 3 vettori:

* bool is\_equal[4] = {false,false,false,false}, che identifica un eventuale taglio per ciascun taglio, già senza dover aggiornare le Vi;
* int Vi\_max[4] = {Vi,Vi,Vi,Vi}, che conterranno alla fine dell’algoritmo i Vi massimi aggiornabili per ogni altezza dell’albero;
* bool tagliato[15] = {false,false,false,false,false,false,false,false,false,false,false,false,false,false,false}, che ci indica se il nodo è stato tagliato da un nodo suo discendente;

1. Si settano le celle is\_equal, se nel piano corrispondente si trova un caso in cui Vs < Vi(con Vs > 0) o Vs = Vi (Vs < 0)
2. Si calcola il massimo delle Vi per ciascun piano e si setta il valore massimo nella cella corrispondente in Vi\_max
3. Si tagliano i figli di quel nodo in cui si verifica che, senza aggiornamento delle Vi, Vs < Vi(con Vs > 0) o Vs = Vi (Vs < 0)
4. Si aggiorna le Vi, cercando, attraverso la funzione di utilità trova\_Vi la Vi corretta, che è sicuramente trovata in un nodo che non è stato ancora tagliato e il cui valore della Vi >= Vs. Se non si trova nessuno valore per cui è vero, si restituisce la posizione precedente. Trovata la Vi giusta, se non siamo nell’indice 0 o 1, possiamo sempre aggiornare e tagliare ipoteticamente quel nodo, se troviamo la prima Vi massima per cui Vi >= Vs. Questo ci consente di eseguire il punto 6 in modo simile a come fa il professore
5. Si stampano i nodi non tagliati con le Vi e Vs correttamente aggiornate, così da poter riuscire a stampare risultati con la convenzione del professore, secondo cui si visitano i nodi da sinistra verso destra e poi dall’alto verso il basso.

Void Costruisci\_albero(vector<int>&ammesso,int pos,int Vi,int \*w,int i)

Questa funzione di utilità è una funzione chiamata dal metodo Branch\_Bound e ci consente di:

1. Quando pos = 0 di allocare dinamicamente un vettore di dimensione 3, che ci servirà a indicare simmetricamente quale indice includere ed escludere, inizializzando tutte le sue celle a -1
2. Si pulisce il vettore ammesso resettando tutti i valori ad 1 che erano rimasti
3. Se pos != 0 e includi\_vet[pos - 1] è false, si evoca la funzione escludi(ammesso,w[i-1]), che esclude nella cella di posizione i-1 di ammesso; altrimenti si evoca la funzione includi(ammesso,w[i-1]), che fa la cosa opposta.
4. Si assegna a w[i] il metodo Vs\_zaino(Vs,ammesso,pos) che chiama l’algoritmo dello zaino binario, considerando tutti i casi già citati nel paragrafo precedente(vedesi le sue prime righe). La funzione restituisce l’indice da includere o escludere nel prossimo passo del Branch And Bound
5. Si aggiorna la posizione pos, tenendo a mente che i figli si trovano nella posizione pos\*2 + 1, e pos\*2+2 per i figli, rispettivamente, sinistri e destri e si fa chiamare ricorsivamente la sua stessa funzione, ma a fine delle sue chiamate si decrementa l’altezza i, finché non è a 0 e, se i != 0 si resetta l’indice da escludere o includere nella posizione i-1(w[i-1] = 0). Analogamente a i, si dimezza la posizione per poter risalire ai nodi padri, tenendo a mente le stesse condizioni poste per l’altezza