1. Esercizi in palestra

Descrizione del problema

Come disse Giovenale "mens sana in corpore sano"; quindi hai deciso di prenderti cura anche del tuo corpo, oltre che della tua mente. Perciò ti sei iscritto in palestra.

Purtruppo non tutti gli esercizi contribuiscono ad aumentare la tua massa muscolare. Alcuni te la possono far perdere. Probabilmente l'istruttore della palestra che ha ideato questi esercizi non è il massimo. Ma hai già pagato in anticipo l'abbonamento annuale.

Decidi di sviluppare il più possibile la tua massa muscolare, eseguendo solamente gli esercizi che ti fanno aumentare la massa e ignorando quelli che te la fanno perdere.

Dettagli

La scheda della palestra contiene N esercizi, ognuno dei quali sviluppa la tua massa muscolare con un grado A_i che indica quanto aumenteranno i tuoi muscoli. Ovviamente nella lista c'è anche qualche esercizio che diminuirebbe la massa.

L'aumento della tua massa muscolare è definito come la somma del grado di aumento della massa degli esercizi eseguiti. Quanto può valere al massimo questa somma?

Assunzioni

- $1 \leq N \leq 10.000$, il numero di esercizi nella scheda.
- $-100 \leq A_i \leq 100$, il grado di aumento di massa dell'i-esimo esercizio.
- È anche possibile che non si svolgano esercizi, in quel caso l'incremento di massa ha valore 0.

Nota bene: se utilizzi il linguaggio **Pascal**, fai attenzione al fatto che il valore massimo contenuto in una variabile `integer` è 32767, troppo piccolo per risolvere questo task completamente. Superando quel numero, infatti, il programma comincerà a salvare numeri imprevedibili (senza mostrarti alcun errore!) per via dell'*overflow*. Per evitare questo fenomeno ti consigliamo di usare **sempre** il tipo `longint` al posto di `integer`.

Dati di input

La prima riga del file di input contiene un intero T, il numero di casi di test. Seguono T casi di test, numerati da 1 a T. Ogni caso di test è preceduto da una riga vuota.

In ciascun caso di test, la prima riga contiene l'unico intero N.

La seconda riga contiene gli N interi separati da spazi, A_i.

Dati di output

Il file di output deve contenere la risposta ai casi di test che sei riuscito a risolvere. Per ogni caso di test che hai risolto, il file di output deve contenere una riga con la dicitura:

`Case #t: k`

dove `t` è il numero del caso di test (a partire da 1) e `k` è il massimo valore di incremento di massa ottenibile.

Esempi di input/output

Input:

```
2

8
1 -4 5 -2 -1 8 0 1

3
-1 -2 -4
```

Output:

```
Case #1: 15
Case #2: 0
```

Spiegazione

Nel **primo caso d'esempio** la soluzione si ottiene eseguendo il primo, il terzo, il sesto, il settimo e l'ottavo esercizio, ottenendo un aumento di massa muscolare di 15.

Nel **secondo caso d'esempio** la soluzione si ottiene non eseguendo alcun esercizio, avendo quindi un aumento di massa muscolare uguale a 0.

Soluzione

```cpp
#include <iostream>
using namespace std;

int main() {
    int T;
    cin >> T;
    for (int t = 1; t <= T; t++) {
        int N; cin >> N;
        int sum = 0;
```

```
        for (int i = 0; i < N; i++) {
            int x;
            cin >> x;
            if (x > 0)
                sum += x;
        }
        cout << "Case #" << t << ": " << sum << endl;
    }
}
```

2. Studenti assetati

Descrizione del problema

È tempo di esami in Luiss. Sei in aula a sostenere l'esame del temuto Prof. Paperon De Paperoni: *Risparmio applicato*. In aula fa particolarmente caldo e tutti gli studenti hanno sete. Dopo molte proteste il Prof. De Paperoni ascolta le suppliche ed accorda il permesso agli studenti di bere per dissetarsi.

Purtruppo vieni scelto tu dal professore per provvedere a questo bisogno della classe. Il professore ti da istruzioni precise: devi spendere il meno possibile per dissetare tutti gli alunni. Con il tempo disponibile puoi solamente recarti al distributore di bibite di fianco all'aula. Spendi il meno possibile, altrimenti il Prof. De Paperoni potrebbe decidere di rimandarti al prossimo appello (e non sarebbe la prima volta!).

Dettagli

Il distributore è un distributore a consumo, è costituito da R rubinetti ed ognuno dei quali eroga una particolare tipologia di bevanda. Ognuna disseta in modo differente: ad esempio per dissetare una persona si hanno bisogno di due bicchieri nel caso di bibita gassata, ed uno nel caso di acqua.

Ogni rubinetto ha un suo costo per ogni erogazione e mostra la quantità di bevanda (in cl) ancora erogabile prima di terminare il proprio serbatoio.

Premendo il tasto di un rubinetto, quest'ultimo eroga una quantità pari ad un bicchiere (sempre della stessa dimensione C in cl) scalando il credito e decrementando di uno i bicchieri a disposizione.

Ogni tipologia B di bevanda nel distributore ha associata una quantità in cl D necessaria per dissetare una persona.

Ogni rubinetto i del distributore è descritto da:

- Q_i: residuo in cl del serbatoio associato
- B_i: tipologia della bevanda al suo interno
- E_i: costo in centesimi di euro di una erogazione

Bisogna dissetare N persone in modo che la somma degli E_i delle erogazioni utilizzate sia minima.

Assunzioni

- Ogni tipologia B_i di bevanda ha una quantità D_i necessaria per dissetare una persona che è sempre multiplo della capacità C del bicchiere
- La capacità residua A_i di ogni rubinetto è sempre un multiplo della quantità D_i necessaria per dissetare una persona con quella bevanda
- È garantito che ci sia sempre abbastanza capacità residua per dissetare tutti
- I fondi a disposizione sono illimitati
- Ogni persona è dissetata allo stesso modo dalle bevande
- $1 \leq N \leq 10000$
- $50 \leq C \leq 500$
- $1 \leq M \leq 100$
- $1 \leq R \leq 100$
- $50 \leq C \leq D_i \leq 500$
- $50 \leq A_i \leq 1000$
- $1 \leq E_i \leq 1000$

Dati di input

La prima riga del file di input contiene un intero T, il numero di casi di test. Seguono T casi di test, numerati da 1 a T. Ogni caso di test è preceduto da una riga vuota.

In ciascun caso di test:

- La prima riga contiene quattro interi, N, C, M, R, separati da uno spazio, che corrispondono rispettivamente: al numero di persone da dissetare, la capacità del bicchiere, il numero di tipologie di bibite presenti nel distributore, il numero di rubinetti presenti nel distributore
- La seconda riga contiene M interi separati da un spazio. L'i-esimo numero corrisponde alla quantità D_i in cl della bevanda i necessaria per dissetare una persona
- Le R righe seguenti descrivono un singolo rubinetto presente nel distributore e contengono ognuna tre interi: A_i, B_i, E_i, separati da uno spazione, che rappresentano rispettivamente: la capacità residua del rubinetto in cl, la tipologia B_i di bevanda erogata e il suo costo E_i in centesimi di euro

Dati di output

Il file di output deve contenere la risposta ai casi di test che sei riuscito a risolvere. Per ogni caso di test che hai risolto, il file di output deve contenere una riga con la dicitura

```
Case #t: s
```

dove `t` è il numero del caso di test (a partire da 1) e il valore `s` corrisponde al numero di centesimi minimi necessari per dissetare tutti gli alunni della classe.

Esempi di input/output

Input:

```
2

5 50 2 8
50 150
300 0 50
150 1 50
150 1 50
150 1 50
50 0 50
300 1 100
100 0 150
1000 0 350

1 100 5 5
200 300 400 100 100
400 0 100
100 3 150
300 1 50
100 4 180
800 2 200
```

Output:

```
Case #1: 250
Case #2: 150
```

Spiegazione

Nel **primo caso d'esempio** una delle alternative più convenienti (non è l'unica) è prendere 5 bicchieri dal rubinetto 0 per un totale di 250 centesimi di euro

Nel **secondo caso d'esempio** l'alternativa più economica (unica in questo caso) è prendere 3 bicchieri dal rubinetto 2 per un totale di 150 centesimi di euro

---

Soluzione

```cpp
#include <iostream>
#include <vector>
#include <array>
```

```cpp
#include <algorithm>
using namespace std;

void solve(int t) {
    // Numero persone, capacità bicchiere, tipologie bibite, numero rubinetti
    int N, C, M, R;
    cin >> N >> C >> M >> R;

    // Bevande = {<tipologia, quantità>}
    vector<int> B(M);

    // Rubinetti = {<capacità, tipologia, costo>}
    vector<array<int, 3>> D(R);

    for (int i = 0; i < M; i++) cin >> B[i];
    for (int i = 0; i < R; i++) cin >> D[i][0] >> D[i][1] >> D[i][2];

    int s = 0;

    vector<array<int, 2>> p;  // <costo, persone>
    for (int i = 0; i < R; i++) {
        // Numero bicchieri
        int nb = D[i][0] / C;
        // Bicchieri per persona
        int cb = B[D[i][1]] / C;
        // Numero persone
        int np = nb / cb;
        // <costo per persona, numero persone>
        p.push_back({cb * D[i][2], np});
    }

    sort(p.begin(), p.end());

    size_t i = 0;
    while (N > 0 && i < p.size()) {
        int r = min(N, p[i][1]);
        s += p[i][0] * r;
        N -= r;
        i++;
    }

    cout << "Case #" << t << ": ";
    cout << s << endl;
}

int main() {
```

```
    int T;
    cin >> T;
    for (int t = 1; t <= T; t++) solve(t);
}
```

3. Produzione di panini

Descrizione del problema

Il bar della Luiss ha deciso di puntare tutto sui suoi deliziosi panini. Sa bene che non avrà difficoltà a vendere tutti i panini che produrrà, quindi cercherà di guadagnare il più possibile con la loro vendita.

Aiuta il bar della Luiss a massimizzare i propri guadagni.

Dettagli

Il bar ha a disposizione n grammi di impasto e m differenti ripieni da utilizzare nei panini. I ripieni sono numerati da 1 a m.

Al bar sono rimasti a_i grammi dell'i-esimo ripieno. Ci vogliono esattamente b_i grammi del ripieno i e c_i grammi di impasto per preparare un panino con l'impasto i. Dopodichè il panino può essere venduto per d_i euro.

È anche possibile preparare panini *senza ripieno*. Ognuno di questi panini vuoti richiede c_0 grammi di impasto e può essere venduto per d_0 euro. Perciò il bar può preparare qualsiasi numero di panini con differente ripieno o senza, a patto che non finisca l'impasto o il ripieno. Dopo aver preparato i panini, eventuali rimanenze vengono buttate perchè non possono essere riutilizzate.

Trova il quantitativo massimo di euro che il bar può guadagnare.

Assunzioni

- $1 \leq n \leq 1000$
- $1 \leq m \leq 10$
- $1 \leq c_0$
- $d_0 \leq 100$
- $1 \leq a_i, b_i, c_i, d_i \leq 100$

Dati di input

La prima riga del file di input contiene un intero T, il numero di casi di test. Seguono T casi di test, numerati da 1 a T. Ogni caso di test è preceduto da una riga vuota.

In ciascun caso di test:

- La prima riga contiene quattro interi, n, m, c_0, d_0, separati da uno spazio, che corrispondono rispettivamente: ai grammi di impasto, il numero di ripieni, la quantità di grammi necessaria per fare un panino vuoto, gli euro che si possono guadagnare dalla vendita di un panino vuoto
- Le m righe seguenti contengono ognuna quattro interi a_i, b_i, c_i, d_i, separati da uno spazio, che corrispondono rispettivamente ai grammi rimasti dell'i-esimo ripieno, i grammi necessari del ripieno i per fare un panino con quel ripieno, i grammi di impasto necessari per produrre un panino con quel ripieno, il numero di euro che si guadagneranno dalla vendita di un panino con quel ripieno

Dati di output

Il file di output deve contenere la risposta ai casi di test che sei riuscito a risolvere. Per ogni caso di test che hai risolto, il file di output deve contenere una riga con la dicitura

```
Case #t: g
```

dove `t` è il numero del caso di test (a partire da 1) e il valore `g` corrisponde al massimo quantitativo di euro che il bar può guadagnare.

Esempi di input/output

Input:

```
2

10 2 2 1
7 3 2 100
12 3 1 10

100 1 25 50
15 5 20 10
```

Output:

```
Case #1: 241
Case #2: 200
```

Spiegazione

Nel **primo caso d'esempio** si produrranno 2 panini con il ripieno 1, 4 panini con il ripieno 2 e 1 panino senza ripieno

Nel **secondo caso d'esempio** si produrranno 4 panini senza ripieno

---

Soluzione

```
#include <iostream>
using namespace std;

const int MAXN = 3100, inf = 1e9;
int N, M;
int a[MAXN], b[MAXN], c[MAXN], d[MAXN];
int w[MAXN], cost[MAXN];
int dp[MAXN][MAXN];

void solve(int t) {
    int cnt = 1;
    cin >> N >> M;
    cin >> c[0] >> d[0];
    for (int i = 1; i <= N; i++) {
        w[cnt] = c[0];
        cost[cnt] = d[0];
        cnt++;
    }
    for (int i = 1; i <= M; i++) {
        cin >> a[i] >> b[i] >> c[i] >> d[i];
        while (b[i] <= a[i]) {
            a[i] -= b[i];
            w[cnt] = c[i];
            cost[cnt] = d[i];
            cnt++;
        }
    }
    cnt--;
    dp[0][0] = 0;
    for (int i = 1; i <= N; i++) dp[0][i] = -inf;
    for (int i = 1; i <= cnt; i++)
        for (int j = 0; j <= N; j++) {
            dp[i][j] = dp[i - 1][j];
            if (j - w[i] >= 0 && dp[i - 1][j - w[i]] + cost[i] > dp[i][j])
                dp[i][j] = dp[i - 1][j - w[i]] + cost[i];
        }

    int ans = 0;
    for (int i = 0; i <= N; i++)
        if (dp[cnt][i] > ans) ans = dp[cnt][i];
    cout << "Case #" << t << ": " << ans << endl;
}

int main() {
```

```
    int T;
    cin >> T;
    for (int t = 1; t <= T; t++) {
        solve(t);
    }
}
```

4. Scambio culturale

Descrizione del problema

La Luiss, all'interno di uno scambio culturale tra atenei italiani e cinesi, ha il compito di organizzare il gemellaggio. Ha compilato una lista dei docenti di entrambe le nazionalità, suddividendoli attraverso una gerarchia di categorie (ad es. due docenti insegnano in una materia, che fa parte di un campo di studi, che fa parte di una facoltà, etc.).

Per rendere lo scambio il più educativo possibile, vuole fare un test facendo lavorare assieme i due docenti che sono più distanti tra loro professionalmente. Questo vuol dire un docente italiano ed uno cinese e che devono avere la distanza massima possibile tra le categorie. Aiuta la Luiss a trovare questi due docenti.

Dettagli

Ci sono n docenti e m categorie etichettate da 1 a m.

Ogni docente i ha una nazionalità N_i che sarà uguale a 0 nel caso di docente italiano e uguale a 1 nel caso di docente cinese.

Ogni docente i appartiene ad una categoria C_j. Inoltre ogni categoria C_i fa parte a sua volta di una categoria C_j. La categoria più generale, che sta a monte di tutte le altre (la categoria "docente"), è etichettata con 0.

La distanza tra due docenti è il numero minimo di passi che si deve fare salendo o scendendo di categoria per arrivare all'altro docente. L'obiettivo è di trovare la lunghezza della distanza massima (tra le distanze minime) di due docenti: uno italiano ed uno cinese.

Assunzioni

- Ogni docente appartiene ad una ed una sola categoria
- Ogni categoria fa parte di una ed una sola altra categoria
- La categoria 0 non fa parte di nessuna categoria
- È garantito che non ci sono cicli all'interno delle categorie
- È garantito che ci siano sempre almeno un docente italiano ed almeno uno cinese
- $10 \leq n \leq 1000$
- $5 \leq m \leq 150$

Dati di input

La prima riga del file di input contiene un intero T, il numero di casi di test. Seguono T casi di test, numerati da 1 a T. Ogni caso di test è preceduto da una riga vuota.

In ciascun caso di test:

- La prima riga contiene due interi, n, m, separati da uno spazio, che corrispondono rispettivamente: al numero dei docenti e al numero delle categorie
- Le n righe seguenti contengono ognuna due interi N_i, C_i, separati da uno spazio, che corrispondono rispettivamente alla nazionalità del docente i ed alla categoria di cui fa parte
- Le m righe seguenti contengono ognuna un intero C_j che corrisponde alla categoria di cui fa parte la categoria j

Dati di output

Il file di output deve contenere la risposta ai casi di test che sei riuscito a risolvere. Per ogni caso di test che hai risolto, il file di output deve contenere una riga con la dicitura

`Case #t: l`

dove `t` è il numero del caso di test (a partire da 1) e il valore `l` corrisponde alla distanza massima tra due docenti di nazionalità differente.

Esempi di input/output

Input:

```
2

8 5
1 2
1 3
1 3
0 3
0 4
0 5
1 5
1 5
0
0
1
1
4
```

```
8 6
0 3
1 3
1 4
0 4
0 5
1 5
0 6
1 6
0
0
1
1
2
2
```

Output:

```
Case #1: 4
Case #2: 4
```

Spiegazione

Nel **primo caso d'esempio** la distanza massima (ma non è l'unica) è tra il docente 0 ed il docente 6

Nel **secondo caso d'esempio** la distanza massima (ma non è l'unica) è tra il docente 7 ed il docente 0

Soluzione

```cpp
#include <vector>
#include <array>
#include <algorithm>
#include <iostream>
using namespace std;

int N, M, sol;
vector<int> C, T;
vector<vector<int>> G;

array<int, 2> dfs(int u) {
    if (u > M) {
        array<int, 2> ret = {-1, -1};
        ret[C[u - M - 1]] = 0;
        return ret;
```

```
    }

    vector<array<int, 2>> c;
    for (int v : G[u]) c.push_back(dfs(v));

    for (size_t i = 0; i < c.size(); i++) {
        for (size_t j = 0; j < c.size(); j++) {
            if (i == j || c[i][0] == -1 || c[j][1] == -1) continue;
            sol = max(sol, c[i][0] + c[j][1]);
        }
    }

    array<int, 2> ret = {-1, -1};
    for (auto& [x, y] : c) {
        if (x != -1) ret[0] = max(ret[0], 1 + x);
        if (y != -1) ret[1] = max(ret[1], 1 + y);
    }

    return ret;
}

void solve(int t) {
    cin >> N >> M;

    C = vector<int>(N);
    T = vector<int>(N + M + 1);
    G = vector<vector<int>>(N + M + 1);

    for (int i = 0; i < N; i++) {
        cin >> C[i] >> T[M + i + 1];
    }

    for (int i = 1; i <= M; i++) {
        cin >> T[i];
    }

    for (int i = 1; i <= M + N; i++) G[T[i]].push_back(i);

    sol = -1;
    dfs(0);

    cout << "Case #" << t << ": " << sol << endl;
}

int main() {
    int T;
```

```
    cin >> T;
    for (int t = 1; t <= T; t++) solve(t);
}
```