Palestra di algoritmi

21/11/2023 - gruppo A

Soluzioni esercizi scorsa volta

Cannoniere

Idea: array in cui salviamo i goal fatti da ogni calciatore usando gli indici

Partiamo da un array di zeri, se abbiamo una riga che indica che il calciatore numero 23 ha fatto 2 goal, sommiamo 2 nella posizione 23 dell'array:

```
for(int i=0;i<N;i++) {
        in >> calciatore >> goal;
        vet[calciatore]+=goal;
}
```

Poi cerchiamo il calciatore con più goal:

```
int massimo = 0;
int indice = 0;
for(int i=0;i<MAX;i++){ //MAX = numero massimo di calciatori (100 in questo caso)
        if(vet[i]>massimo) {
                massimo = vet[i];
                indice = i;
        }
}
out << indice << " " << massimo;
```

Halloween candies

Idea: ordiniamo i team per punteggio crescente e aumentiamo di 1 le caramelle ogni volta

```
sort(vet, vet+N);
long long int caramelleDaDare = 1;
long long int caramelleTotali = 1;
for(int i=1;i<N;i++){
	if(vet[i-1]!=vet[i]){
		caramelleDaDare++;
	}
	caramelleTotali+=caramelleDaDare;
}
out << caramelleTotali;
```

Quadrati

Idea: al posto di controllare se un numero è un quadrato perfetto, partiamo dalla base e eleviamo finché siamo nel range

```
in >> a >> b;
x = sqrt(b);    //troviamo la base di inizio
y = sqrt(a);    //troviamo la base di fine
if (y*y!=a)     //se estremo iniziale escluso
      n = x-y;
else            //se estremo iniziale incluso
      n=x-y+1;
out << n;
```

Taxi

Idea: problema greedy per eccellenza. Ad ogni distributore scelgo se mi conviene continuare col taxi che sto usando o se cambiare

```
int N, attuale, prezzo;
int spesaTotale = 0;

in >> N >> attuale;
spesaTotale = attuale;

for(int i=1;i<N;i++) {
        in >> prezzo;
        attuale++;                  //ogni volta che faccio uno spostamento, il prezzo aumenta
        if(prezzo<attuale)          //se mi conviene cambiare
                attuale = prezzo;
        spesaTotale+=attuale;
}

out << spesaTotale;
```

Disk failure 2

Idea: problema molto semplice di implementazione, dobbiamo solo controllare di avere abbastanza tempo

```
int sera, mattina, N, tempo;
int ris = -1;

in >> N >> tempo >> mattina; //non ci interessa quando inizia il primo contest

for(int i=1;i<N;i++) {
        in >> sera >> mattina;
        if(((24-sera) + mattina)>=tempo) {  //se ho abbastanza tempo
                ris = i-1;
                break;                      //istruzione per fermare il ciclo
        }
}
out << ris;
```

Isogram

Un po’ complesso a livello di implementazione, ma lo vediamo assieme.

Idea: per ogni stringa, fare la stessa cosa di cannoniere, ma con i vari caratteri, ossia salvarsi quante volte un carattere appare e controllare che siano tutte minori o uguali di 2

```
string a;
int count = 0;              //numero di quasi-isogrammi totali
int v[MAX];                 //vettore per contare quante volte compare un carattere
getline(in, a);             //salvo una riga in a
int n = stoi(a);            //trasformo la stringa in int

for(int i=0;i<26;i++) {     //inizializzo vettore dei caratteri
        v[i]=0;
}
```

```
for(int i=0;i<n;i++) {
        string s;
        getline(in,s); //prendo in input la riga
        bool ok = true;

         for(int j=0;j<s.size() && ok;j++) {
                if(isalpha(s[j])) {                 //ritorna true se il carattere è una lettera
                        s[j] = toupper(s[j]);       //metto tutte le lettere in maiuscolo per uniformare
                        int c = s[j] - 'A';                 //trovo l'indice della lettera
                        if(v[c]==2) {                       //se il carattere era già stato ripetuto due volte
                                        ok = false;
                        } else {
                                        v[c]++;
                        }
                }
        }
        if(ok) {count++;}
        for(int i=0;i<30;i++) { v[i]=0; }           //inizializzo di nuovo il vettore
}

out << count;
```

Ricorsione

Ricorsione

Risolvere un problema dividendolo in sottoproblemi.

Ogni sottoproblema:

- ha la struttura identica al problema originale
- è più piccolo di quello originale

Esempio 1 - sommatoria

Vogliamo sommare i numeri da 1 a n.

Se per esempio n=5 vogliamo fare 1+2+3+4+5.

Si può risolvere con un semplice ciclo for, lo so, ma dobbiamo partire da qualcosa di semplice per arrivare a casi più complessi in cui serve la ricorsione per forza.

Esempio 1 - sommatoria

Possiamo dividere il problema in:

- parto da n (5 nel nostro caso) e lo aggiungo alla mia variabile somma
- risolvo il problema sui numeri rimasti (quindi partendo da n-1=4), fin quando non finisco i numeri

La ricorsione in codice

Ma come si fa tutto ciò in codice?

In c++ possiamo scrivere funzioni che chiamano sé stesse.

Esempio:

```
void stampaCiao(){
	cout << “ciao\n”;
	stampaCiao();
}
```

Questa funzione stampa ciao all’infinito, perchè continua a chiamare sé stessa senza mai fermarsi.

La ricorsione - punti chiave

Per scrivere una funzione ricorsiva quindi:

- dobbiamo risolvere un problema che può essere scomposto in problemi più piccoli
- ci devono essere dei “casi base”, ossia delle condizioni per cui la ricorsione si ferma, senza continuare all’infinito

Nel nostro esempio, se partiamo dal 5, la ricorsione si deve fermare a 0, perchè non c’è più nulla da sommare, se non 0.

Se partissimo da 0 invece dovrebbe fermarsi una volta che arriva a n, perché non vogliamo sommare i numeri successivi.

Esempio 1 - sommatoria - codice

```
int sommatoria(int n) {
	if(n<=0) { //ho finito i numeri da sommare
		return 0;
	} else{
		return n+sommatoria(n-1); //ritorno n + la somma di tutti gli altri numeri
	}
}
```

Nel main: cout << sommatoria(5);

Esempio 2 - fattoriale

```
int fattoriale(long long int n){
        if(n<=1) return 1;
        else return n*fattoriale(n-1);
}
```

Nel main: cout << fattoriale(10)

Praticamente uguale alla sommatoria

Esempio 3 - Fibonacci

La serie di Fibonacci è una serie di numeri in cui i primi due sono 1, mentre tutti gli altri sono dati dalla somma dei due numeri precedenti.

In forma di funzione:
Fibonacci(1) = 1
Fibonacci(2) = 1
Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-1)

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,144, 233, 377, 610, 987,1597, 2584, 4181, …

Esempio 3 - Fibonacci - codice

```
int fibonacci(int n) {
	if(n==1 || n==2) {
		return 1;
	} else if(n>2) {
		return fibonacci(n-1) + fibonacci(n-2);
	}
}
```

Esercizio

Scrivere una funzione ricorsiva che stampi tutte le possibili stringhe di numeri binari di lunghezza n.

Esempio:

n = 3

000
001
010
011
100
101
110
111

Soluzione

```
void fun(string s, int n) {
	if(n==0) {
		cout << s << "\n";
	} else {
		string s1 = s+'0';
		string s2 = s+'1';
		fun(s1, n-1);
		fun(s2, n-1);
	}
}
```

Più difficile

Riscrivere il programma di prima, ma stavolta avete in input due numeri n0 e n1. Le stringhe binarie che stampate possono avere al massimo n0 zeri consecutivi e n1 uni consecutivi.

Esempio: n=6, n0=2, n1=3

Il programma deve stampare (non per forza in quest'ordine):

```
100100 010100 110100 001100 101100 011100 010010
110010 001010 101010 011010 111010 100110 010110
110110 001110 101110 001001 101001 011001 111001
100101 010101 110101 001101 101101 011101 010011
110011 001011 101011 011011 111011 100111 010111
110111
```

```
// fun(“”, 6, 2, 3, 0, 0);
void fun(string s, int n, int n0, int n1, int messi0, int messi1) {
	if(n==0) {
		cout << s << "\n";
	} else {
		if(messi0<n0) {
			string s1 = s+'0';
			fun(s1, n-1, n0, n1, messi0+1, 0);
		}
		if(messi1<n1) {
			string s1 = s+'1';
			fun(s1, n-1, n0, n1,0, messi1+1);
		}
	}
}
```

Problemi

- domino (https://training.olinfo.it/#/task/domino/statement)
- borse di studio (https://training.olinfo.it/#/task/oii_borse/statement)