

ASD Laboratorio 08

The A(SD)-Team

UniTN

2024-03-21

CALENDARIO

27/02	Programmazione dinamica
25/03	Programmazione dinamica
06/05	Algoritmi approssimati 1
15/05	Algoritmi approssimati 2
26/05	Presentazione Progetto 2
27/05	Progetto alg approssimati
29/05	Progetto alg approssimati

PROGETTO ALGORITMI APPROSSIMATI

- Il secondo progetto verrà assegnato il **23/05/2024** e avrete circa una settimana di tempo. Il progetto riguarderà gli algoritmi approssimati (ultima parte del corso);
- Per iscriversi (se non vi siete iscritti al primo progetto):
https://bit.ly/ASDprog_2023-2024 (dovete essere loggati con l'account UniTN)
- Assumiamo gli stessi gruppi del primo semestre, in caso di cambiamenti, avvisare **entro il 19/05/2024**;

SOTTOSEQUENZA CRESCENTE

Data una sequenza di interi scegliere un sottoinsieme della sequenza in modo che:

- gli elementi del sottoinsieme, messi nell'ordine in cui si trovavano nella sequenza originaria, formino una sequenza crescente
- il sottoinsieme abbia somma massima

SOTTOPROBLEMA

$S(i)$ = somma della sottosequenza crescente di somma massima a partire dall'elemento i

Non funziona! Per scegliere ottimamente, abbiamo bisogno di sapere l'ultimo elemento scelto.

SOTTOPROBLEMA

$S[i, j]$ = somma ottimale ottenibile dal sottoarray $[i..N - 1]$ avendo scelto per ultimo l'elemento j

$$S[i, j] = \begin{cases} 0, & \text{if } i == n \\ S[i + 1, j], & \text{if } A[i] < A[j] \\ \max(S[i + 1, j], S[i + 1, i] + A[i]) & \text{if } A[i] \geq A[j] \end{cases}$$

SOTTOPROBLEMA ALTERNATIVO

$S[i]$ = somma ottimale da i in poi essendo obbligati a scegliere l'elemento i

$$S[i] = A[i] + \max_{j:(j>i, A[j]\geq A[i])}(S[j])$$

La soluzione del problema è uguale a $\max(S)$.

Funzione di ricorrenza ($v[i]$: valore dell' i -esimo elemento, $p[i]$: peso dell' i -esimo elemento)

$S(c, i)$ = massimo valore ottenibile utilizzando gli elementi da i in poi, con uno zaino avente spazio c .

$$S(c, i) = \begin{cases} -\infty & \text{if } c < 0 \\ 0 & \text{if } i == N \\ \max \begin{cases} v[i] + S(c - p[i], i + 1) \\ S(c, i + 1) \end{cases} & \text{if } i < N \end{cases}$$

```
int ric(int c,int i){
    if(c<0)
        return -100000000;
    if(i==N)
        return 0;
    int p=elements[i].first;
    int v=elements[i].second;
    return max(v+ric(c-p,i+1),
              ric(c,i+1));
}
```

Nota: l'ordine dei casi base è importante.

ZAINO MEMOIZATION

```
int ric(int c,int i){
    if(c<0)
        return -100000000;
    if(i==N)
        return 0;
    if(sav[c][i]==-1){
        int p=elements[i].first;
        int v=elements[i].second;
        sav[c][i]= max(v+ric(c-p,i+1),
                       ric(c,i+1));
    }
    return sav[c][i];
}
```

ZAINO ITERATIVO

- Il calcolo di $S(c, i)$ dipende dagli $S(c', i + 1)$.
- Calcoliamo prima tutti gli $S(_, N - 1)$, poi tutti gli $S(_, N - 2)$...

```
for(int i=N-1;i>=0;i--){
    int p=elements[i].first;
    int v=elements[i].second;
    for(int c=0;c<=C;c++){
        if(elements[i].first<=c)
            sav[c][i]=max(sav[c][i+1],
                          v+sav[c-p][i+1]);
        else
            sav[c][i]=sav[c][i+1];
    }
}
```

ZAINO ITERATIVO EFFICIENTE

- Una volta calcolati tutti gli $S(_, i)$, gli $S(_, i + 1)$ non ci servono più.
- Utilizziamo un array $C \cdot 2$.

```
for(int i=N-1;i>=0;i--){
    int p=elements[i].first;
    int v=elements[i].second;
    int cur=i%2; int next=(i+1)%2;
    for(int c=0;c<=C;c++){
        if(elements[i].first<=c)
            sav[c][cur]=max(sav[c][next],
                           v+sav[c-p][next]);
        else
            sav[c][cur]=sav[c][next];
    }
}
```

PILLOLE

$S[i, j]$ = numero di combinazioni ottenibili da una bottiglia contenente i pillole intere e j pillole smezzate

$$S[i, j] = \begin{cases} 1, & \text{if } i == 0 \text{ and } j == 0 \\ S[i - 1, j + 1], & \text{if } i > 0 \text{ and } j == 0 \\ S[i, j - 1], & \text{if } i == 0 \text{ and } j > 0 \\ S[i, j - 1] + S[i - 1, j + 1], & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

PROBLEMI (I)

SOTTOSEQUENZA COMUNE MASSIMALE

Date due stringhe di caratteri alfanumerici, calcolare una sottosequenza comune massimale (secondo la definizione delle slides di Montresor). Stamparne la lunghezza.

Letture di una stringa (libreria string):

```
string s;  
in>>s;
```

Ottenere dimensione stringa e valore per un singolo carattere:

```
int dim=s.size();  
char c=s[2];
```

PROBLEMI (II)

DEFINIZIONE: NODE-COVER

Un insieme $S \subseteq V$ di nodi è un Node-Cover se ogni arco nel grafo/albero ha almeno uno dei due nodi in S .

MIN COVER SU ALBERO

Dato un albero, trovare la dimensione del Node-Cover di dimensione minima.

MIN COVER SU ALBERO PESATO

Dato un albero con pesi su i nodi, trovare il Node-Cover di peso minimo e stamparne il peso.

LA VENDETTA DEL RE LICH

Progetto programmazione dinamica a. a. 2016/2017

ASSEDIO A NASSAU

Progetto programmazione dinamica a. a. 2017/2018