ASD Laboratorio 08

The A(SD)-Team

UniTN

2026-04-09

Calendario

27/02	Programmazione dinamica
09/04	Programmazione dinamica
28/04	Algoritmi approssimati 1
07/05	Algoritmi approssimati 2
14/05	Presentazione Progetto 2 (lezione)
19/05	Progetto alg approssimati
21/05	Progetto alg approssimati

Secondo progetto

Progetto algoritmi approssimati

- Il secondo progetto verrà assegnato **giovedì 14/05/2026** e avrete circa una settimana di tempo. Il progetto riguarderà gli algoritmi approssimati (ultima parte del corso);
- Iscrizione dei gruppi al progetto entro **domenica 03/05/2026** (nuova data!):

  `https://bit.ly/ASDprog_2025-2026_2` (dovete essere loggati con l'account UniTN)
- Iscrivetevi al contest ECIU sempre entro **domenica 03/05/2026**:

  `https://bit.ly/ASDprog_2025-2026_ECIU` (dovete iscrivervi singolarmente, i gruppi sono definiti tramite il form di cui sopra)

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.

Sottosequenza crescente

Sottoproblema

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

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

Sottosequenza crescente

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 $\max(S)$.

Zaino

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.

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

ZAINO RICORSIVO

```
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 efficente

- 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.

Lettura 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.

Progetti degli anni passati

La vendetta del Re Lich

Progetto programmazione dinamica a. a. 2016/2017

Assedio a Nassau

Progetto programmazione dinamica a. a. 2017/2018