Programmazione dinamica

Corso per le Olimpiadi Italiane di Informatica

Slide di Giovanni Spadaccini

Overview del corso

1. Prerequisiti, I/O e Complessità
2. Vettori e Greedy
3. Ricorsione
4. **Programmazione Dinamica**
5. Grafi

Dividiamo il corso in moduli chiari e strutturati per una preparazione completa.

Risorse e Materiale di Studio

- Libro di riferimento: Guida Quinta Edizione
- Esercizi online per pratica e approfondimenti. OII
- Documentazione C++ e Python
- Di più alla fine

Cosa è la Programmazione Dinamica?

- **Definizione:** Un metodo per risolvere problemi complessi decomponendoli in sottoproblemi più semplici.
- **Utilizzo:** Ottimale per problemi di ottimizzazione e decisionali.
- **Approcci:** Ricorsivo, con memoizzazione, e con tabulazione.

Perché è importante nella programmazione competitiva?

- Fornisce soluzioni efficienti a problemi che altrimenti richiederebbero un tempo computazionale eccessivo.
- Di solito è un problema delle territoriali.

Approcci alla Programmazione Dinamica

- **Ricorsivo:** Partire da un caso base e costruire la soluzione step by step.
- **Memoizzazione:** Salvare i risultati di sottoproblemi per evitare calcoli ripetitivi.
- **Tabulazione:** Costruire una tabella per calcolare i risultati in modo iterativo, partendo dai casi base verso le soluzioni dei problemi più grandi.

Esempi di Proprietà della Programmazione Dinamica

1. **Sottoproblemi Sovrapposti**
   - Esempio: Sequenza di Fibonacci, in cui il calcolo di `F(n)` richiede `F(n-1)` e `F(n-2)`.
2. **Struttura Ottimale**
   - Una soluzione ottimale globale può essere costruita dalle soluzioni ottimali dei suoi sottoproblemi.

1. Qual è il caso base?

- Identificate il caso più semplice che può essere risolto senza calcoli.

2. Quali sono gli stati ripetuti?

- Capire quali stati siano già stati calcolati

3. Come si può passare da uno stato all'altro?

- Definire la relazione di ricorrenza tra gli stati.

Esempio: Sequenza di Fibonacci

- **Definizione:** `F(n) = F(n-1) + F(n-2)`, con `F(0) = 0` e `F(1) = 1`.
- **Caso Base:** `F(0) = 0` e `F(1) = 1`.
- **Stati Ripetuti:** disegnando il tree possiamo vedere che ricalcoliamo molte volte gli stessi stati (adesso è molto evidente poiché c'è una sola variabile).
- **Relazione di Ricorrenza:** `F(n) = F(n-1) + F(n-2)`.

Codice

```c
int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

- Disegno del tree per `fib(5)`

Memoizzazione

- Osserviamo come fin(n) è calcolato più volte, e come lo il risultato di fib(n) è lo stesso per ogni chiamata.

```
#define MAXN 100
int fib_mem[MAXN];

int fib(int n) {
    if (n <= 1) return n;
    if (fib_mem[n] != -1) return fib_mem[n];
    return fib_mem[n] = fib(n-1) + fib(n-2);
}

void init() {
    //memset(fib_mem, -1, sizeof(fib_mem));
    for (int i = 0; i < MAXN; i++) fib_mem[i] = -1;
}
```

Introduzione al Problema dello Zaino

Il problema dello zaino è un problema di decisione di ottimizzazione che mira a massimizzare il valore totale degli oggetti inseriti in uno zaino, rispettando il limite di peso.

Definizione del Problema

- **Input:**
  - Un insieme di `n` oggetti, ognuno con un peso `w[i]` e un valore `v[i]`.
  - Un peso massimo `W` che lo zaino può portare.
- **Obiettivo:** Massimizzare il valore totale degli oggetti nello zaino senza superare il peso `W`.

Versione ricorsiva del Problema TOP-DOWN

```
// N il numero degli oggetti
struct oggetto{
 int peso;
 int valore;
};
oggetto oggetti[N];
int knapsack_ricorsivo(int n) {
    if (n==0) return 0;
    int max = 0;
    for (int i = 0; i < N; i++) {
        int preso = 0;
        if (n - oggetti[i].peso >= 0)
            preso = oggetti[i].valore + knapsack_ricorsivo(n - oggetti[i].peso);
        if (preso > max)
            max = preso;
    }
    return max;
}
```

versione iterativa Bottom-Up

```
struct oggetto{
int peso;
int valore;
};

oggetto oggetti[100];
int soluzioni[1000];

int knapsack_bottom_up(int n) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j <= M - oggetti[i].peso ; j++)
            if (soluzioni[j] + oggetti[i].valore > soluzioni[j+oggetti[i].peso])
                soluzioni[j+oggetti[i].peso] = soluzioni[j] + oggetti[i].valore;
    return soluzioni[M];
}
```

versione ricorsiva Top-Down con memorizzazione

```
int top_down(int n) {
    if (soluzioni[n] != -1) return soluzioni[n];
    int max = 0;
    for (int i = 0; i < N; i++) {
        int preso = 0;
        if (n - oggetti[i].peso >= 0)
            preso = oggetti[i].valore + knapsack_top_down(n - oggetti[i].peso);
        if (preso > max)
            max = preso;
    }
    soluzioni[n] = max;
    return max;
}
```

versione knapsack 1-0

di ogni oggetto si può prendere solo una copia

Inizializzazione della Tabella di Programmazione Dinamica

Creiamo una matrice `dp` dove `dp[i][j]` rappresenta il valore massimo che può essere raggiunto con i primi `i` oggetti e un limite di peso `j`.

```
int dp[n+1][W+1];
for (int i = 0; i <= n; i++) {
    for (int j = 0; j <= W; j++) {
        if (i == 0 || j == 0) dp[i][j] = 0;
    }
}
```

Codice per il Riempimento della Tabella

```
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= W; j++)
        if (w[i-1] <= j)
            dp[i][j] = max(dp[i-1][j], dp[i-1][j-w[i-1]] + v[i-1]);
        else
            dp[i][j] = dp[i-1][j];
```

Analisi della Soluzione

- La soluzione al problema è contenuta in `dp[n][W]`, che darà il valore massimo ottenibile senza superare il peso `W`.

```
print(f"Il valore massimo ottenibile è: {dp[n][W]}")
```

Complessità e Considerazioni

- **Complessità Temporale:** O(nW), dove `n` è il numero di oggetti e `W` è il peso massimo.
- **Complessità Spaziale:** O(nW) per lo spazio utilizzato dalla tabella `dp`.
- **Ottimizzazione:** Possibili riduzioni dello spazio se manteniamo solo la riga corrente e la precedente.

Esempio Con ottimizzazione dello spazio

```
int dp[W+1][2];
for(int i=0;i<=W;i++){
    dp[i][0]=0;
    dp[i][1]=0;
}
for(int i=1;i<=n;i++){
    for(int j=1;j<=W;j++){
        if(w[i-1]<=j)
            dp[j][i%2]=max(dp[j][1-i%2],dp[j-w[i-1]][1-i%2]+v[i-1]);
        else
            dp[j][i%2]=dp[j][1-i%2];
    }
}
```

La dieta di poldo

Un altro esempio di programmazione dinamica è la dieta di poldo

Il dottore ordina a Poldo di seguire una dieta. Ad ogni pasto non può mai mangiare un panino che abbia
un peso maggiore o uguale a quello appena mangiato

In verità il problema è `Longest increasing subsequence`

soluzione dynamic programming

```
int soluzioni[MN];
int panini[MN];

int main(int argc, char *argv[]) {
    fstream in,out;
    int numeroPanini,max;
    in.open("input.txt",ios::in);
    out.open("output.txt",ios::out);
    in >> numeroPanini;
    int max;
    for (int i=0;i<numeroPanini;i++)
        in >> panini[i];
    for (int i=numeroPanini-1;i>=0;i--) {
        max=0;
        for (int j=numeroPanini-1;j>i;j--) {
            if (panini[i] > panini[j] && soluzioni[j]>max)
                max=soluzioni[j];
        }
        soluzioni[i] = max + 1;
    }
    max=soluzioni[0];
    for (int i=1; i< numeroPanini;i++)
        if (soluzioni[i] > max)
            max = soluzioni[i];
    out << max;
}
```

soluzione overkill

```
#define MN 10000
int dp[MN];
int ar[MN];
int main(){
    int n;
    cin >>n;
    for(int i=0;i<n;i++){cin >>ar[n-i-1];} //memorizzo al contrario
    int mi=1;
    dp[0]=ar[0];//dp terra solo i valori crescenti
    for(int i=1;i<n;i++){
        int *tmp=lower_bound(dp,dp+mi,ar[i]);//cerco il primo elemento maggiore o uguale
        int d=tmp-dp;//calcolo la distanza
        if(d==mi){
            dp[mi]=ar[i];
            mi++;
        }else dp[d]=ar[i];
    }

    cout <<mi<<endl;
}
```

Loot Box

lootbox

```cpp
#define rep(i,n) for(int i=0;i<n;i++)
#define fep(i,j,n) for(int i=j;i<n;i++)
#define ay array
#define ii ay<int,2>
#define MN 100001
#define ML 5000

ii ar[ML];
int dp[MN];

int main(){
    int n,k;
    cin >>n>>k;
    rep(i,n)
        cin >>ar[i][1]>>ar[i][0];
    sort(ar,ar+n);
    int ans=0;
    rep(i,n){
        int costo=ar[i][0];
        for(int c=k-costo;c>=0;c--){
            dp[c+costo]=max(dp[c]+ar[i][1],dp[c+costo]);
            ans=max(dp[c+costo],ans);
        }
    }
    printf("%d",ans);
}
```

Treni

treni

```cpp
#include <bits/stdc++.h>

using namespace std;

#define ll long long
#define F first
#define S second
#define ay array
#define PB push_back
#define rep(i, n) for (int i = 0; i < n; ++i)
#define MN 1000002

int dp[MN];

int tempo_massimo(int N, int a[], int b[])
{
    for(int i=1;i<=N;i++){
        dp[i+1]=max(dp[i]+a[i-1],dp[i-1]+b[i-1]);
    }


    return dp[N+1];
}
```

Esercizio: Multicore

multicore

```cpp
#include <cstdio>
#include <fstream>
#include <iostream>
#include <climits>
#include <string>

#define MAXN 300
#define MAXC 200

using namespace std;

int main() {

  // freopen("input.txt", "r", stdin);
  // freopen("output.txt", "w", stdout);

  int T, t, i;
  cin >> T;

  for (t = 1; t <= T; t++) {
    int N, B;
    int C[MAXN], P[MAXN];

    int nc=0;

    cin >> N >> B;
    for (i = 0; i < N; i++)
      cin >> C[i] >> P[i];

    int dp[MAXC*MAXN+2][2];
    for(int i = 0; i < MAXC*MAXN+2; i++) dp[i][0] = dp[i][1] = INT_MAX;

    dp[0][0] =  dp[0][1] = 0;

    for(int i = 0; i < N; i++) {
      for(int j = 0; j <= (MAXC*MAXN+2); j++) {
        dp[j][i%2] = dp[j][(i+1)%2];
        if(j >= C[i] && dp[j-C[i]][(i+1)%2] != INT_MAX){
          int price = dp[j-C[i]][(i+1)%2] + P[i];
          if(price<=B){
            nc = max(nc, j);
            dp[j][i%2] = min(dp[j][i%2],price );
          }
        }
      }
    }
    cout << "Case #" << t << ": " << nc << endl;
  }
}
```

Altro esempio di memorizzazione su grafi

Un altro caso di ricorsiva, ropes

Problema: Dato un albero con `n` nodi, ogni nodo ha un peso `w` (e un genitore). Dobbiamo unire l'albero in una sola corda, tagliando i nodi. Il costo di taglio di un nodo è `w`, e il costo di unione è `0`. Trovare il costo minimo per unire l'albero.

```cpp
#include <bits/stdc++.h>

using namespace std;
vector<array<int,2>> node[100001];
int arr[100001];
int dfs(int a){
    if(node[a].size()==0)return 0;
    int ma=0;
    int sum=0;
    for(int i=0;i<node[a].size();i++){
        sum+=dfs(node[a][i][0]);
        sum+=min(ma,node[a][i][1]); // questa riga fa in modo che il massimo non sia mai preso
        ma=max(ma,node[a][i][1]);
    }
    return sum;
}
int main(){
    int n;
    cin >>n;
    for(int i=0;i<n;i++)
        cin >> arr[i];
    for(int i=0;i<n;i++){
        int w;
        cin >> w;
        node[arr[i]].push_back({i+1,w});
    }
    cout <<dfs(0);
}
```

Livello preparazione nazionali dp

Musical Notes

In verità in questo problema la dynamic programming viene utilizzata parzialmente ma è lo stesso un bel problema.

```cpp
#include <bits/stdc++.h>
using namespace std;
#define MN 100000
class SegmentTree {
private:
    vector<int> tree;
    int n;
    void build(int node, int start, int end) {
        if (start == end) {
            tree[node] = 0;
        } else {
            int mid = (start + end) / 2;
            build( 2*node+1, start, mid);
            build( 2*node+2, mid+1, end);
            tree[node] = max(tree[2*node+1], tree[2*node+2]);
        }
    }

    void update(int idx, int val, int node, int start, int end) {
        if (start>0 && end>0 && start>end) return;
        if (start == end) {
            tree[node] = val;
        } else {
            int mid = (start + end) / 2;
            if (start <= idx && idx <= mid) update(idx, val, 2*node+1, start, mid);
            else update(idx, val, 2*node+2, mid+1, end);
            tree[node] = max(tree[2*node+1], tree[2*node+2]);
        }
    }

    int query(int L, int R, int node, int start, int end) {
        if (R < start || end < L) return 0;
        if (L <= start && end <= R) return tree[node];
        int mid = (start + end) / 2;
        int left_query = query(L, R, 2*node+1, start, mid);
        int right_query = query(L, R, 2*node+2, mid+1, end);
        return max(left_query, right_query);
    }

public:
    SegmentTree(int n) {
        this->n = n;
        tree.resize(4*n);
        build( 0, 0, n-1);
    }
    void update(int idx, int val) { update(idx, val, 0, 0, n-1); }
    int query(int L, int R) { return query(L, R, 0, 0, n-1); }
};

int main() {
  int n;
  cin >> n;
  vector<int> v(n);
  for (auto &x : v) {
    cin >> x;
    x--;
  }
  vector<int> comp(n);
  for (auto &c : comp) {
    cin >> c;
    c--;
  }
  SegmentTree st(n);
  int last_comp[n];
  for (int i = 0; i < n; i++) {
    last_comp[i] = 0;
  }
  int res = 0;
  last_comp[comp[v[0]]] = 1;
  st.update(v[0], 1);

  for (int i = 1; i < n; i++) {
    int last_before = st.query(0, v[i] - 1);
    int last = max(last_before, last_comp[v[i]])+1;
    res = max(res, last);
    last_comp[comp[v[i]]] = max(last_comp[comp[v[i]]], last);
    st.update(v[i], last);
  }
  cout << res << endl;
}
```

Struttura delle gare

Di solito:

- un es greedy
- un es stringe/sorting
- un es dp/sorting
- un es grafi

Materiali per approfondimenti

- Libro di riferimento: Guida Quinta Edizione
- Lezioni tenute da ragazzi di codefarm: https://www.youtube.com/playlist?list=PLxSVZC2doc7cxBBLqoMlCjOd6MeMFRnbl
- Se volte sperimentare con i problemi che troverete alla gara https://territoriali.olinfo.it/

Materiali avanzati (livello nazionali)

molto comprensiva di materiali (anzi troppo)

- Buon modo per iniziare e guardare diversi argomenti: https://algobadge.olinfo.it/
- Libro di riferimento gratis (in inglese): https://cses.fi/book/book.pdf
- Tutti le implementazioni di tutti i più svariati algoritmi: https://cp-algorithms.com/
- Principale sito per i Competitive Programmers codeforces.com
- Buon modo per iniziare anche se molto pi`u complicato : https://usaco.guide/
- Corso avanzato codefarm: https://www.youtube.com/playlist?list=PLxSVZC2doc7cxBBLqoMlCjOd6MeMFRnbl