Dynamic Programming

Find the solution of a large instance by finding and efficiently memorizing the solutions of small instances.

Ex.

Finding the n-th Fibonacci number.

Fibonacci numbers: 1 1 2 3 5 8 13 21 ...

$f(0) = f(1) = 1, f(n) = f(n-1) + f(n-2)$, for all $n>=2$.

Python function

```
def fib_DP(n):
    a, b = 1, 1
    for i in range(2, n+1):
        a, b = b, a+b
    return b
```

Notice the difference of a recursive call

```
def fib_RC(n):
    if n<=1:
        return 1
    else:
        return fib_RC(n-1) + fib_RC(n-2)
```

Demo: fib.py

10 mm

Methods for IP : branch-and-bound, dynamic programming (DP) etc.

(DP)

Ex. 0-1 Knapsack problem

Given n items with size $a_i > 0$
value $c_i > 0$

1 container capacity $b > 0$

Object: pack the items so that the total value is maximized.

formulation

$$\max \quad Z = \sum c_i x_i$$

$$\text{s.t.} \quad \sum a_i x_i \leq b$$

$$x_i \in \{0,1\}$$

Here, we assume $a_i, b, c_i \in \mathbb{Z}$ (integer)

$$\max \quad 3x_1 + 4x_2 + x_3 + 2x_4$$

$$\text{s.t.} \quad 2x_1 + 3x_2 + x_3 + 3x_4 \leq 4$$

$$x_i \in \{0,1\}$$

A "simple" yet difficult problem

enumeration method $\Rightarrow O(n2^n)$ time.

DP

Let $f(i,k) = \max\limits_{\substack{x_i \in \{0,1\} \\ \sum_{j=1}^{i} a_j x_j \le k}} \sum_{j=1}^{i} c_j x_j \qquad i = 1, 2, \cdots, n \quad k = 0, 1, \cdots, b$

Then $f(1,k) = \begin{cases} 0 & k < a_1 \\ c_1 & k \geqslant a_1 \end{cases}$

and

$$f(i,k) = \max\left\{ f(i-1,k),\ f(i-1,\ k-a_i) + c_i \right\},\quad i \geqslant 2$$

$(\text{assume } f(i-1, k-a_i) = -\infty \text{ if } k < a_i)$

Ex.

k i	1	2	3	4
1	0	3	3	3
2	0	3	4	4
3	1	3	4	5
4	1	3	4	⑤

optimal

running time $O(nb)$

better if $b < 2^n$.

結構導向的方法

Shortest Path Problem

Input: Graph G=(V,E), edge length l(u,v), s, t
Output: a shortest s-t path (or its nonexistence)

Note: it may not exist if there exists a negative cycle.

In the following, we assume there is no such a cycle.

Method 1: find the shortest one from ALL paths

=> Too many paths! (see Movie 1)

Method 2: Pulling method (=> Dijkstra's method)

=> Movie 2

Bellman-Ford algo for the shortest path problem

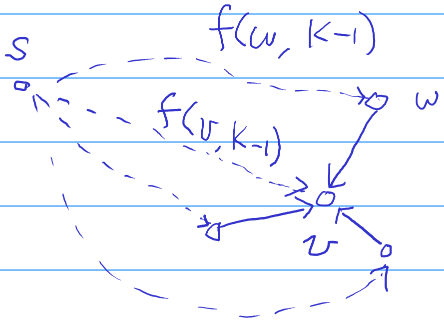
Define

$f(v, k)$ = length of a shortest s-v path that uses at most k edges

$\Longrightarrow$ We want $f(v, n-1)$ for all v.

$$f(v,0) = \begin{cases} 0 & v = s \\ \infty & v \neq s \end{cases}$$

$$f(v,k) = \min\left\{ f(v,k-1), \min_{w:(w,v)\in E} \{ f(w,k-1) + \ell(w,v) \} \right\}$$



Observation

We can safely drop the second parameter in f(v, k), i.e., consider

$$f(v) = \min\left\{ f(v), \min_{w:(w,v)\in E} \{ f(w) + \ell(w,v) \} \right\}.$$

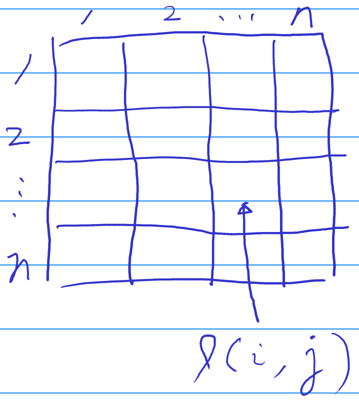
$\Longleftrightarrow$ Bellman-Ford algo

** Advanced topic (optional)

- Dijkstra's algorithm: 1-1 or 1-many/all
- Bellman-Ford algorithm: 1-all
- Floyd-Warshall: all-all

More efficent than n times of the first two.

Let V = {1, 2, ..., n}, and we use the incidence matrix.



```
main

    for i = 1, 2, ..., n
        for j = 1, 2, ..., n

            dist[i, j] = $\begin{cases} 0 & i = j \\ l(i,j) & (i,j) \in E \\ \infty & \text{otherwise} \end{cases}$

    for k = 1, 2, ..., n
        for i = 1, 2, ..., n
            for j = 1, 2, ..., n
                if dist[i, j] > dist[i, k] + dist[k, j] {
                    dist[i, j] = dist[i, k] + dist[k, j]
                }
```

time: O(n^3)
space: O(n^2)

Correctness

f(i, j, k) = length of a shortest i-j path that uses only nodes 1, ..., k

$$\Rightarrow \quad f(i, j, 0) = \begin{cases} 0 & i = j \\ l(i,j) & (i,j) \in E \\ \infty & \text{otherwise} \end{cases}$$

$$f(i, j, k) = \min\{f(i, j, k-1), f(i, k, k-1) + f(k, j, k-1)\}$$