DYNAMIC PROGRAMMING

1) Longest path in a DAG $G=(V,E)$

Subproblem $L(v)=$ length of longeth path ending in v

Recurrency $L(v)=\max_{(u,v)\in G}(L(u)+1)$ (where an empty max equals 0)

Order to compute topological sorted order of G

Run Time: $O(|V|+|E|)$

2) Longest Increasing Subsequence of sequence $a_1,\dots,a_n$

Subproblem: $L(i)=$ length of longest increasing subsequence ending in a_i

Recurrency: $L(i)=1+\max_{j<i} L(j)\,\mathbb{1}_{a_j<a_i}$

Order to compute: $i=1,2,3,\cdots$

Run Time: $O(n^2)$

3) Edit Distance of two strings $x[1:n]$, $y[1:m]$

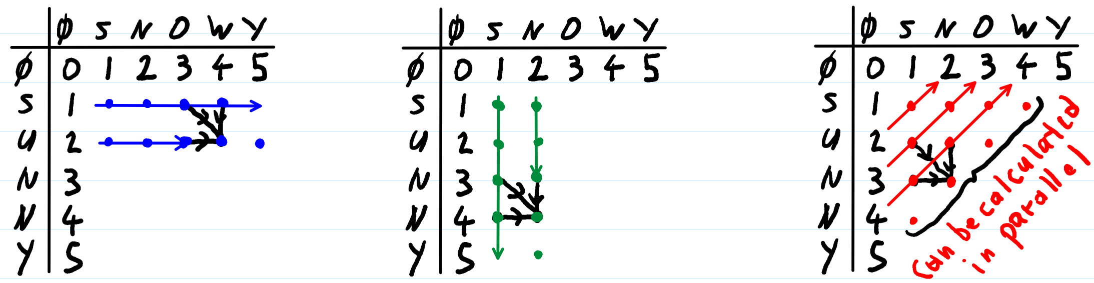
$E(x[1:n],y[1:m]) = \min\left\{\begin{array}{l}\#\text{ of deletes, inserts, substitutes}\\ \text{to transform } x[1:n] \text{ into } y[1:m]\end{array}\right.$

Subproblem: $E(x[1:i],y[1:j])$

Recurrency: $E(x[1:i],y[1:j])=$

$$=\min\begin{cases}E(x[1:i],y[1:j-1])+1\\ E(x[1:i-1],y[1:j])+1\\ E(x[1:i-1],y[1:j-1])+\mathbb{1}_{x_i\neq y_j}\end{cases}$$

Order to compute:



Run Time	$O(nm)$	$O(nm)$	$O(n+m)$ in parallel
space	$O(m)$	$O(n)$	$O(n+m)$

4) Knapsack

Input: Capacity W (integer)

weights $w_1, \cdots, w_n$ (integers)

values $v_1, \cdots, v_n$ (")

Goal: Find set of item with

max total value, with total weight $\le W$

4a) Knapsack with Replacement

We looked optimal solution

$$v_{i_1} + \cdots + v_{i_k} = v_{i_1} + \cdots + v_{i_{k-1}} + v_{i_k} = K(W - w_{i_k}) + v_{i_k}$$

Subproblem:

$K(C)$ = max total value with total weight $\le C$

$C = 0, 1, \cdots, W$

Recurance:

$$K(C) = \max_{i : w_i \le C} \left(v_i + K(C - w_i) \right)$$

Order to compute

$C = 0, 1, \cdots$, starting with $K(0) = 0$

Algorithm:

Input: W, $v[1:n]$, $w[1:n]$

$K(0) = 0$

For $C = 1$ to W

$$K(C) = \max_{i: w_i \leq C} v_i + K(C - w_i)$$

Output: $K(W)$

Runtime: $O(nW)$

space: $O(W+n)$

exponential in $\log W$

4b) Knapsack w/o replacement

1) Subproblem

Optimal solution: items $i_1, \cdots i_k$, $\sum w_{i_\ell} \leq W$

$$\text{Value} = v_{i_1} + \cdots + v_{i_k} = \underbrace{v_{i_1} + \cdots + v_{i_{k-1}}}_{\text{Guess: } \tilde{K}(W - w_{i_k})} + v_{i_k}$$

$\tilde{K}(C)$ optimum w/o replacement w/ capacity C

Problem: $i_1, \cdots i_{k-1}$ must be different from i_k, but in $\tilde{K}(W - w_{i_k})$ i_k could be used again

2nd Try:

$$\tilde{K}(k) = \text{Knapsack}(v_1, \cdots v_k, w_1, \cdots w_k)$$

$$\tilde{K}(n) = \max\{\tilde{K}(n-1), v_n + \tilde{K}(n-1)\}$$

Problem: We are not keeping track of the budget

3rd Try:

$K(C, k)$ = Optimum with total weight $\leq C$ using a subset of items $1, \cdots, k$

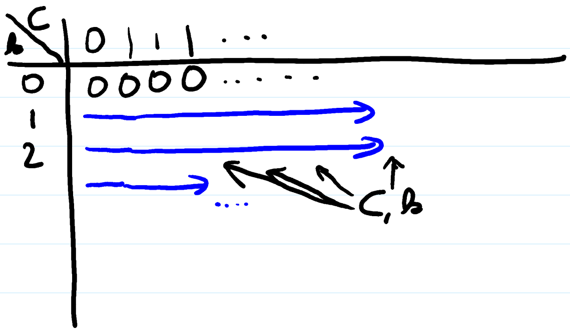
2) Recursion

$K(C, k-1)$ if $v_k > C$

2) Recursion

$$K(C,k) = \begin{cases} K(C,k-1) & \text{if } v_k > C \\ \max\{K(C,k-1),\ K(C-w_k,k-1)+v_k\} & \text{if } v_k \le C \end{cases}$$

3) Order



$W(C,0) = 0$

Algorithm:

> Input: W, $v[1:n]$, $w[1:n]$
>
> For $c = 0,1,\cdots W$
>
> $K(C,0) = 0$
>
> For $k = 1,\cdots n$ "Calculate $W(k,\cdot)$"
>
>   For $C = 1,\cdots W$
>
> $K(C,k) = K(C,k-1)$
>
>   If $w_k \le C$ and $v_k + K(C-w_k,k-1) > K(C,k-1)$
>
> $K(C,k) = v_k + K(C-w_k, k-1)$
>
> Output $K(W,n)$

Runtime

$O(nW)$

space

$\cancel{O(nW)}$ $O(n+W)$

better implementation

5) Single Source Shortest Path

Input: Weighted graph $G=(V,E,\ell)$, source s (where $\ell(u,v)\in\mathbb{R}$)

Output: $\forall v \in V$, $\mathrm{dist}(v)$ = length of shortest path $s \rightsquigarrow v$

1) Subproblems

Optimal solution:

$v_1 = s, v_2, \cdots, v_k = v$

Optimal solution:

$v_1 = s_1, v_2, \cdots, v_k = v$

$dist(v) = dist(v_{k-1}) + \ell(v_{k-1}, v)$

$\Rightarrow dist(v) = \min_u (dist(u) + w_{uv})$ not really a subproblem!

Idea: recurse on # of edges k in path

$dist(v, k) =$ length of shortest path $s \rightsquigarrow v$ using k edges

$k = 0, 1, \cdots$

Recursion

$$dist(v, k) = \min \left\{ dist(v, k-1), \min_{uv \in E} \left(dist(u, k-1) + \ell(u,v) \right) \right\}$$

Dependencies

$dist(v, k)$ depends on $dist(u, k-1)$ for $uv \in E$

Algorithm

```
For v ∈ V : dist(v,0) = ∞
dist(v,0) = 0
For k = 1, ..., n-1
For all v ∈ V
    dist(v, k) = dist(v, k-1)
    For all (u,v) ∈ E
        If dist(v,k) > dist(u, k-1) + ℓ(u,v)
            dist(v,k) = dist(u, k-1) + ℓ(u,v)
∀v∈V Output dist(v, n-1)
```

Run time

$O(|V|(|E| + |V|)) \cong O(|E||V|)$

Rem: This is very similar to Bellman Ford, in fact, in the worst case, it is identical. (sometimes Bellman Ford just needs one run, e.g. if $G =$ , in which case B.F. will be done after one run through all edges, while D.P. needs n-1 runs

6) All Pairs Shortest path

Input: $G=(V,E)$ $\ell: E \to \mathbb{R}$

Output: $dist(u,v)$ = length of shortest path from u to v for all $u,v \in V$

Idea I: Run Bellman-Ford $|V|$ times Run Time $O(|V|^2|E|)$

Today: DP Solution Floyd Warshall — " — $O(|V|^3)$

Optimal Solution:

$$dist(u,v) = \ell(u,v_1) + \ell(v_1,v_2) + \cdots + \ell(v_k, v)$$

Idea 1:

$$dist(u,v,k)$$

k ← # of edges used = $\underbrace{\text{\# of edges}}_{\text{\# of intermediate vertice } +1}$ used

Problem: by iteration on the number of intermediate edges, we effectively run the algorithm from 5 for all sources s, getting again $O(|V|^2|E|)$ run-time.

Idea 2:

Iterate on which vertices are used

Subproblem

$V = \{1,2,\cdots n\}$

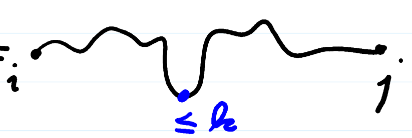
$dist(i,j;k)$ uses intermediate vertices $\leq k$

Base Case

$$dist(i,j;0) = \ell(i,j)$$

Recursion:

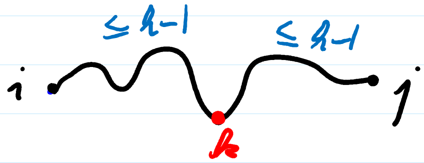
Assume $dist(i,j,k-1)$ is known

$dist(i,j,k) =$ 

Case 1: k is not used

$= dist(i, j, k-1)$

Case 2: k is used



$= ?$

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

Algorithm:

$FW(G, \ell)$

> For $i, j = 1, \cdots n$ $\quad dist(i,j,0) = \infty$
> For $(i,j) \in E$ $\quad dist(i,j,0) = \ell(i,j)$
> For $i = 1, \cdots n$ $\quad dist(i,i,0) = 0$
> For $k = 1, \cdots, n$
>   For $i = 1, \cdots, n$
>   For $j = 1, \cdots, n$
> $dist(i,j,k) = \min\{dist(i,j,k-1), dist(i,k,k-1) + dist(k,j,k-1)\}$
>
> Output $dist(i,j,n)$

7) Travelling Salesman Problem (TSP)

Given: n cities, distances d_{ij} $\quad i \neq j$

Goal: Find path of minimal length,

starting at 1, ending at 1, visiting every city once

Naive Running Time $\quad n! = O((\frac{n}{e})^n)$

Goal: Find optimal tour in time $O(2^n)$

Idea: Consider subproblem on

set of cities $S \subseteq \{1, \cdots n\}$, $1 \in S$

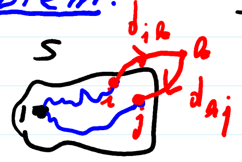
Idea 1:

$C(S) \quad$ = cost of tour in S starting

Idea 1:

$C(S)$ = cost of tour in S, starting and ending in 1

Problem: $S \to S \cup \{k\}$



??

Idea 2:

$$C(S) = \min_{j \neq 1} \text{Cost}(\text{Tour } 1 \to j \text{ in } S) + d_{j \to 1}$$

Subproblem:

Let $1, j \in S$, $j \neq 1$

$C(S, j)$ = Length of shortest path from 1 to j visiting every $i \in S$ once