Chapter 3. Algor. Design Techniques

We discuss 3 techniques:

- Divide_and. Conquer
- Dynamic Programming
- Greedy Method

3.1. The Divide_and. Conquer Technique.

We illustrate this technique through a few examples:

- Multiplying long integers
- Finding minimum/maximum
- Faster integer multiplication

Basic Idea:

- Divide probl. into smaller subprobl.
- Recursively solve subproblems
- Combine solutions of subprobl. into a solution of larger probl.

Multiplying Long Integers

The problem is def. as follows.

Input. 2 n-bit integers X, Y.

Output. The product $X * Y$.

Remark. School method takes $\Theta(n^2)$ steps.

Goal: An $O(n^{\lg 3})$ algor. based on divide-and-conquer.

Idea:

$X =$ | A | B | $= A * 2^{n/2} + B$

$Y =$ | C | D | $= C * 2^{n/2} + D$

Then:

$$X * Y = A*C*2^n + (A*D + B*C)2^{n/2} + B*D$$

Based on this, we have the recurrence:

$T(1) = 1$ and $T(n) = 4T(n/2) + cn$

which has solution: $T(n) = O(n^2)$

Q: Can we reduce 4 multiplications of $n/2$-bit integers to 3?

Consider this: (Making use of $A*C$, $B*D$)

$$X * Y = A*C*2^n + \{(A-B)*(D-C) + A*C + B*D\} * 2^{n/2} + B*D$$

⇒ We reduce 1 $n/2$-bit integer multipl. at the expense of 4 $n/2$-bit integer additions.

The recurrence becomes:

$$T(1) = 1 \quad , \quad T(n) = 3\,T(n/2) + c.n$$

whose solution is:

$$T(n) = O(n^{\lg 3}) = O(n^{1.59})$$

Finding Maximum & Minimum of a Set.

A straightforward approach:

- Find max using n-1 comparisons
- Find min of remaining n-1 elements using n-2 comparisons

⇒ Total of $2n-3$ comparisons

The divide-and-conquer approach:

- Find max & min of $A[1..n/2]$, $A[n/2+1,n]$
- Now find max of 2 max's
  min of 2 min's
  using 2 comparisons.

Thus: $T(2) = 1$, $T(n) = 2T(n/2) + 2$

Solution is: $T(n) = \frac{3}{2}n - 2$

Pf. $T(n) = 2 + 2T(n/2)$

$= 2 + 2\{2 + 2T(n/4)\}$

$(n = 2^k)$ $= 2 + 4 + 8 + \cdots + 2^{k-1} + 2^{k-1}T(2)$

$= \sum_{i=1}^{k-1} 2^i + 2^{k-1}$

$= 2\left(\sum_{i=0}^{k-2} 2^i\right) + 2^{k-1}$

$= 2(2^{k-1} - 1) + 2^{k-1}$

$= 2^k + 2^{k-1} - 2$

$= n + \frac{n}{2} - 2$

$= \frac{3n}{2} - 2$ □

Fast Divide-and-Conquer Integer Multiplicat.

Observation. In previous divide-and-conquer integer multiplication we reduce 4 $\frac{n}{2}$-bit int. mult. to 3 $\frac{n}{2}$-bit int. mult. and achieve $O(n^{\log_2 3})$

Note: $a = 3$, $b = 2$ $\Rightarrow$ $O(n^{\log_2 3})$.

Question. Can we get a faster alg. by splitting x, y into even smaller pieces?

Let's revisit previous algor.

$$X = \boxed{X_1 \mid X_2} = X_1 \cdot 2^{\frac{n}{2}} + X_2$$

$$Y = \boxed{Y_1 \mid Y_2} = Y_1 \cdot 2^{\frac{n}{2}} + Y_2$$

Setting $t = 2^{\frac{n}{2}}$ we rewrite

$$X(t) = X_1 \cdot t + X_2 \quad ; \quad Y(t) = Y_1 \cdot t + Y_2$$

$\Rightarrow$ $X(t)$, $Y(t)$ are polynomials in t.

$$\Rightarrow Z(t) := X(t) \cdot Y(t)$$

$$= \underbrace{X_1 Y_1}_{a} \cdot t^2 + \underbrace{(X_1 \cdot Y_2 + X_2 \cdot Y_1)}_{b} t + \underbrace{X_2 \cdot Y_2}_{c}$$

$$= a \cdot t^2 + b \cdot t + c$$

$Z(t)$ is a polynomial in t of degree 2.

Note: a, b, c are coefficients of $Z(t)$

Question. Can we compute a, b, c fast?

Fact. For a polynomial of degree l, if we know its values at $l+1$ points, then its coefficients can be det. through interpolation using La Grange's formula.

Ex. Consider $Z(t) = X(t) \cdot Y(t)$ and the points $t = -1, 0, 1$. We have:

$$Z(t) = \sum_{i=-1}^{1} \prod_{\substack{j=-1 \\ j \neq i}}^{1} \frac{t-j}{i-j} \cdot Z(i)$$

$$= \boxed{\begin{aligned} &\frac{t}{-1} \cdot \frac{t-1}{-1-1} \cdot Z(-1) \\ &+ \frac{t+1}{1} \cdot \frac{t-1}{-1} \cdot Z(0) \\ &+ \frac{t+1}{2} \cdot \frac{t}{1} \cdot Z(1) \end{aligned}} = \boxed{\begin{aligned} &\frac{t^2-t}{2} Z(-1) \\ &+ \frac{-t^2+1}{1} \cdot Z(0) \\ &+ \frac{t^2+t}{2} Z(1) \end{aligned}}$$

$$= \underbrace{\left(\frac{1}{2} Z(-1) - Z(0) + \frac{1}{2} Z(1)\right)}_{a} \cdot t^2 + \underbrace{\left(-\frac{1}{2} Z(-1) + \frac{1}{2} Z(1)\right)}_{b} \cdot t + \underbrace{Z(0)}_{c}$$

Observation. Given $Z(0), Z(-1), Z(1)$ the coefficients a, b, c of $Z(t)$ can be computed via additions & subtractions.

Observation. If we split X, Y into k pieces, then $X(t), Y(t)$ are of degree $k-1$ and $Z(t) = X(t) \cdot Y(t)$ is of degree $2k-2$.

Given fixed k, choose suitable small values for the $2k-1$ points $t_1, \ldots, t_{2k-1}$:

(1) Evaluation: Compute $X(t), Y(t)$ on the $2k-1$ points $t_1, \ldots, t_{2k-1}$.
(Since $t_1, \ldots, t_{2k-1}$ are fixed, theirs powers t_i^j's are fixed. Hence, $X(t), Y(t)$ can be computed using simple add/sub. of $\frac{n}{k}$-bit integers.)

(2) Point-wise Product. Compute $Z(t) = X(t) \cdot Y(t)$ for $t = t_1, \ldots, t_{2k-1}$.
(This req. $2k-1$ mult. of $\frac{n}{k}$-bit int.)

(3) Interpolation. Using La Grange's formula, compute the coefficients of $Z(t)$ $\left(= \sum_{j=0}^{2k-2} \underbrace{\left[\sum * Z(t_i) \right]}_{\hookrightarrow \text{ from Lagrange}} t^j \right)$

Complexity of Step (1):

$$X(t) = X_{k-1} t^{k-1} + \cdots + X_1 t + X_0$$

$$Y(t) = Y_{k-1} t^{k-1} + \cdots + Y_1 t + Y_0$$

X_j, Y_j are $\frac{n}{k}$-bit integers, $0 \le j \le k-1$

t_i^j's are fixed

$\Rightarrow X(t_i), Y(t_i)$ can be evaluated in

$$O(k \cdot k \cdot \frac{n}{k} \underbrace{k \cdot \log k}_{\text{length of } t_i^j}) = O(n)$$

(i, j)

since k is fixed.

Step (2): Note that mult. of $\frac{n}{k} + O(k \log k)$-bit integers is $T(\frac{n}{k}) + O(n)$, k is const.

Complexity of Step (3):

Recover $z_{2k-2}, \ldots, z_0$ from

$z(t_1), \ldots, z(t_{2k-1})$

based on La Grange's formula

For $j = 0, \ldots, 2k-2$:

$$z_j = \sum z(t_i) \qquad i = 1, \ldots, 2k-1$$

takes $O(k \cdot \frac{n}{k}) = O(n)$

(Note: addition of $\frac{n}{k} + O(k \log k)$-bit integers can be done in $O(n)$ where k is const.)

Step (1):

$1 \le i \le 2k-1$, $0 \le j \le k-1$

$X_j * t_i^j$ takes $O(\frac{n}{k} \cdot k \cdot \log k)$ bit operations each

($\frac{n}{k}$ bits, $k \log k$ bits)

$\Rightarrow O(k \cdot k \cdot \frac{n}{k} k \cdot \log k) = O(n)$

to compute these terms

$X(t_i) = \sum_{j=0}^{k-1} X_j t_i^j$ takes $O(k \cdot \frac{n}{k} \cdot k \cdot \log k) = O(n)$

$Y(t_i)$ is similar

$\Rightarrow$ Step (1) takes $O(n)$

Step (2):

$X(t_i) * Y(t_i)$ take $T(\frac{n}{k}) + O(n)$ steps

($\frac{n}{k} + c$) bits ($\frac{n}{k} + c$) bit

Thus, the recurrence is:

$$T(n) = (2k-1) \cdot T\left(\frac{n}{k}\right) + O(n)$$

$$\Rightarrow T(n) = O\left(n^{\log_k (2k-1)}\right).$$

Note: $T(n) = O(n^{1+\varepsilon})$ when k is sufficiently large, where $\varepsilon > 0$

(Remark: The integers in (2) may be slightly longer than $\frac{n}{k}$ bits. However, operations can be done on $\frac{n}{k}$-bit int. and the extra cost can be absorbed into the term $O(n)$.)

Note: For a k^{th} degree polynomial $Z(t)$ La Grange's formula for $t = -\frac{k}{2}, \ldots, 0, \ldots, \frac{k}{2}$:

$$Z(t) = \sum_{i=-\frac{k}{2}}^{\frac{k}{2}} \cdot \prod_{\substack{j=-\frac{k}{2} \\ j \neq i}}^{\frac{k}{2}} \left(\frac{(t-j)}{(i-j)} \cdot Z(i) \right)$$

3.2. Dynamic Programming

We've seen a few examples: NFAs → REs, CYK algor. in Automata Theory

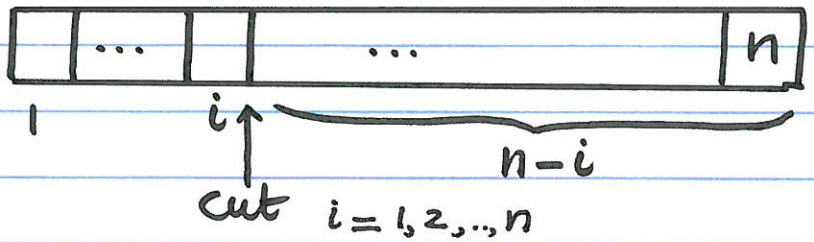
Rod Cutting

Input. A rod of length n and sale prices p_i for rod of length i, $i = 1, \ldots, n$.

Output. Max. revenue r_n by cutting rod and selling the pieces.

Fact. There are 2^{n-1} ways of cutting rod of length n into pieces.

Pf.



Cut at position i → rod of length i and recursively cut rod of length $n-i$
Now prove Fact by induction on n. □

Observation. From above proof we have:

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

p_i: sale price of rod of length i

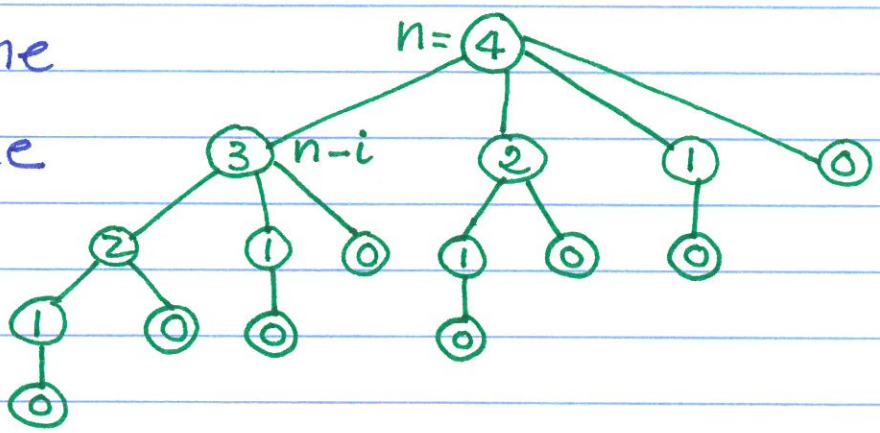
r_{n-i}: max rev. from rod of length $n-i$

Given array $p[1..n]$ of sale prices and n:

CUT-Rod (p, n)

```
if n = 0 return 0
q := -∞
for i = 1 to n
    q = max(q, p[i] + CUT-Rod(p, n-i))
return q
```

This gives the recursion tree (for $n = 4$)



Running time of Cut-Rod is:

$$T(n) = O(n) + \sum_{j=0}^{n-1} T(j) = O(2^n)$$

Observ. There are repititions in rec. tree of CUT-Rod. For ex., if we have computed r_2 before, we don't have to re-comp. it.

$\Rightarrow$ Store interm. results in $r[0..n]$ & proceed recursively from smaller subproblems.

```
Revised-Cut-Rod(p,n)
  Let r[0..n] be new
  r[0] = 0                    /* r[j] = Max_{1≤i≤j} {p[i]+r[j-i]}
  for j=1 to n
      q = -∞
      for i=1 to j
(*)       q = max(q, p[i]+r[j-i])
      r[j] = q
  return r[n]
```

Example.

i	1	2	3	4	5	6	7	8	9	10
p[i]	1	5	8	9	10	17	17	20	24	30

i	0	1	2	3	4	5	6	7	8	9	10
r[i]	0	1	5	8	10	13	17	18	22	25	30

Question. How to obtain the opt. cut?

Solution. In line (*) record value $\hat{i}$ for which the max is found using array $S[0..n]$.

```
(**)  if q < p[i] + r[j-i] then
          q = p[i] + r[j-i]
          s[j] = i
```

Example.

i	0	1	2	3	4	5	6	7	8	9	10
r[i]	0	1	5	8	10	13	17	18	22	25	30
s[i]	0	1	2	3	2	2	6	1	2	3	10

Principles of Dynamic Prog.

- Compute solutions to all subproblems: starting from smaller subproblems and proceeding to larger.
- Store solutions to subproblems in a table (programming ↔ tabular nature)
- Once a subproblem is solved, its solution is stored in table. (no need to recalculate)
- At most poly. many subprobl. of size $< n$ each solvable in polynomial time.

World Series Odds

- Two teams A, B play $\leq 2n-1$ games
- First team achieving n victories is WINNER

Assumptions: (1) No ties

(2) Game outcomes are independent

(3) $P(A \text{ wins}) = p$

$P(B \text{ wins}) = q = 1-p$

$P(i,j) =$ Probability A eventually wins the series given that
- A needs i more victories to win
- B " j " " "

Thus, $P(n,n)$ = Prob. A wins before the entire match.

$$P(0,i) = 1 \quad , \quad 1 \le i \le n$$
$$P(i,0) = 0 \quad , \quad 1 \le i \le n$$

We have: For $i,j \ge 1$:

$$P(i,j) = p \cdot P(i-1,j) + q \cdot P(i,j-1)$$

Goal: Compute $P(i,j)$, $i,j \ge 0$

```
W-SERIES (i,j)
    if i = 0 then return 1
      else if j = 0 then return 0
    return p . W-SERIES(i-1,j) + q . W-SERIES(i,j-1)
```

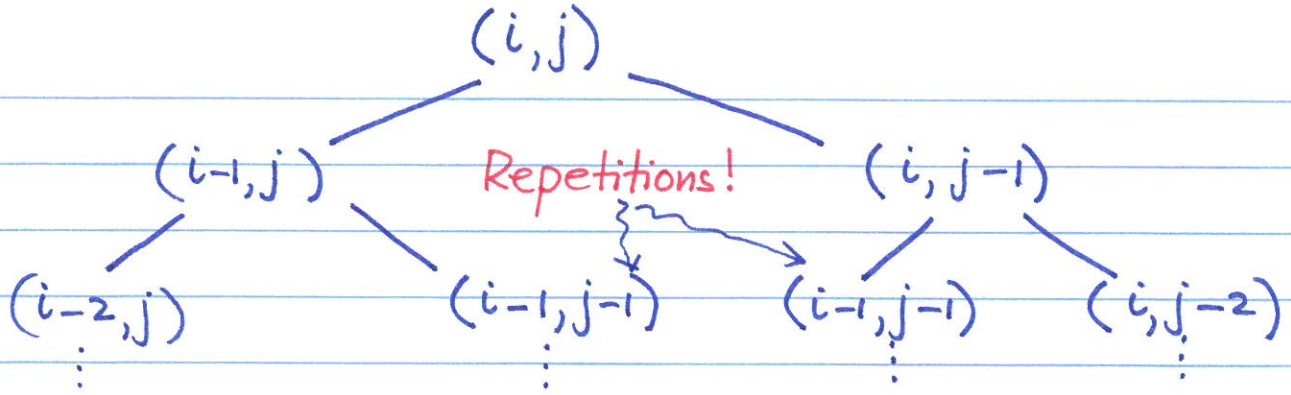
Let $T(k)$, $k = i+j$, denote running time of W-SERIES(i,j). Then

$$T(1) = \Theta(1) \text{ and } T(k) \le 2T(k-1) + \Theta(1)$$

Thus, $T(k) = O(2^k) = O(4^n) \mid_{k=2n}$

Claim. $T(k) = \Omega(4^n/n)$ $\quad i=j=n$

Pf. What does W-SERIES' recursion tree look like?



Total number of nodes is $2\binom{i+j}{j} - 1$

For $i = j = n$, noting $\binom{2n}{n} \geq \frac{4^n}{2n+1}$ we have:

$$T(2n) = \Omega\left(\binom{2n}{n}\right) = \Omega\left(\frac{4^n}{2n+1}\right)$$

$$= \Omega\left(\frac{4^n}{n}\right) \quad \boxtimes$$

Observ. There are only n^2 values to be computed, and they can be stored in a table!

Ex. Table of odds $(p = q = \frac{1}{2}, n = 4)$

j \ i	0	1	2	3	4
4	1	15/6	13/6	21/32	1/2
3	1	7/8	11/16	1/2	11/32
2	1	3/4	1/2	5/16	3/16
1	1	1/2	1/4	1/8	1/16
0		0	0	0	0

← upper half!

lower half!

⇒ Calculate lower half to table and then upper half!

```
REV. W. SERIES (n,p)
    Let P[0..n, 0..n] be array
    q = 1-p
    for s = 1 to n do        /* lower half */
        P[0,s] = 1 ; P[s,0] = 0 ;
        for k = 1 to s-1 do
            P[k,s-k] = p.P[k-1,s-k] + q.P[k,s-k-1]
    /* We Now calculate upper half */
    for s=1 to n do
        for k=0 to n-s do
            P[s+k, n-k] = p.P[s+k-1, n-k] + q.P[s+k,
                                                n-k-1]
```

Time complexity : $O(n^2)$

Chained Matrix Multiplication

We wish to compute matrix product

$$M = M_1 \times M_2 \times \cdots \times M_n$$

Different orders of calculation require different number of arith. operations.

Ex. $M = A \times B \times C \times D$

A is 13×5, B is 5×89, C is 89×3, D is 3×34

(1) $((AB)C)D$ requires 10,582 scalar mult.

(Note: $A \times B$ req. 5,785 " "

$(A \times B) \times C$ — 3,471 " "

$((A \times B) \times C) \times D$ — 1,326 " "

(2) $(AB)(CD)$ requires 54,201 " "

(3) $(A(BC))D$ — 2,856 " "

(4) $A((BC)D)$ — 4,055 " "

(5) $A(B(CD))$ — 26,418 " " □

Goal: To compute the opt. order that gives the least number of scalar mult. in computing the product $M_1 \times \cdots \times M_n$

Question. How many ways can we parenthesize $M_1 \times M_2 \times \cdots \times M_n$?

Let $S(n)$ be # of ways to parenthesize $M_1 \times \cdots \times M_n$. Then $S(1) = 1$ and

$$S(n) = \sum_{i=1}^{n-1} S(i) \cdot S(n-i)$$

Claim. $S(n) = \frac{1}{n}\binom{2n-2}{n-1} = \Omega\left(\frac{4^n}{n^2}\right)$ □

Let M_i be of dimension $d_{i-1} \times d_i$, $i = 1, \ldots, n$

Let m_{ij} denote the opt. number of scalar mult. for $M_i \times \cdots \times M_j$.

Then: $m_{ii} = 0 \qquad i = 1, \ldots, n$

$$m_{i,i+1} = d_{i-1} \times d_i \times d_{i+1} \qquad i = 1, \ldots, n-1$$

and for $1 < s < n$ and $i = 1, \ldots, n-s$

$$m_{i,i+s} = \min_{i \le k < i+s} \left(m_{ik} + m_{k+1,i+s} + d_{i-1} d_k d_{i+s} \right)$$

$(M_i \times \cdots \times M_k)(M_{k+1} \times \cdots \times M_{i+s})$

$d_{i-1} \times d_k \rightarrow m_{ik}$ $\qquad$ $d_k \times d_{i+s} \rightarrow m_{k+1,i+s}$

Observ. We only need to compute $O(n^2)$ values m_{ij}'s which can be stored in a table!

4	$m_{14} = 2856$			
3	$m_{13} = 1,530$	$m_{24} = 1,845$		
2	$m_{12} = 5,785$	$m_{23} = 1,335$	$m_{34} = 9,878$	
1	$m_{11} = 0$	$m_{22} = 0$	$m_{33} = 0$	$m_{44} = 0$

↑

```
MAT-MULT ( d[0..n], n)
  Let m[1..n, 1..n] be an array
  for i = 1 to n do  m[i,i] = 0
  for s = 1 to n-1 do
      for i = 1 to n-s do
          j = i+s
          m[i,j] = min_{i≤k<j} ( m[i,k] + m[k+1, j]
                                 + d_{i-1} * d_k * d_j )
```

Time complexity : $O(n^3)$

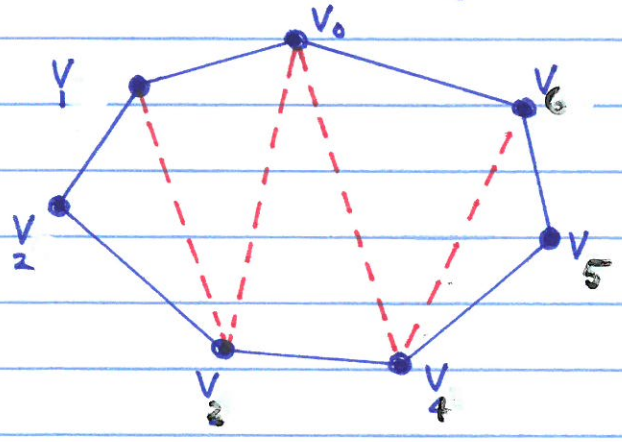
(Note the similarity between this and the CYK algorithm.)

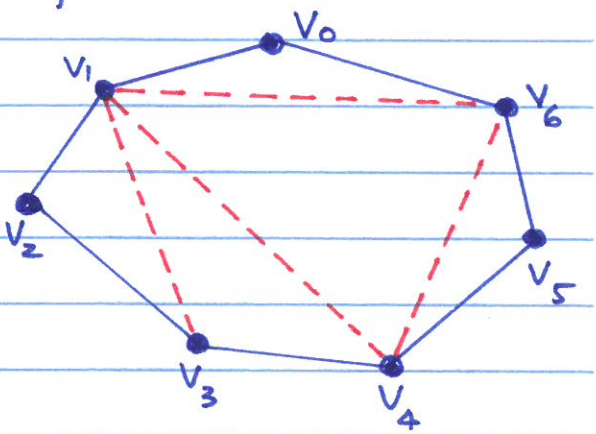
Question. How to output the optimal order?

The Triangulation Problem.

- A polygon is represented by listing its vertices in counterclockwise order
- If $P = \langle v_0, v_1, \ldots, v_{n-1} \rangle$ is a polygon, then its sides are: $\overline{v_0 v_1}, \overline{v_1 v_2}, \ldots, \overline{v_{n-1} v_n}$ (where $v_n = v_0$)
- If v_i, v_j are nonadjacent, $\overline{v_i v_j}$ is chord
- A chord divides a polygon into 2 polygons.
- A triangulation T is a set of chords that divides P into disjoint triangles.

Ex. Two triangulations of a polygon





Note: In a triangulation: no chords intersect one another and any other chord intersects a chord in the triangulation.

Fact. Every triangulation of an n-vertex polygon has n-3 chords and yields n-2 triangles. □

Polygon triangulation is def. as follows:

Input. A polygon $P = \langle v_0, v_1, \ldots, v_{n-1} \rangle$ and a weight function ω def. on all triangles $\omega(\Delta v_i v_j v_k)$.

Output. A triangulation T that minimizes the sum of weights of its triangles.

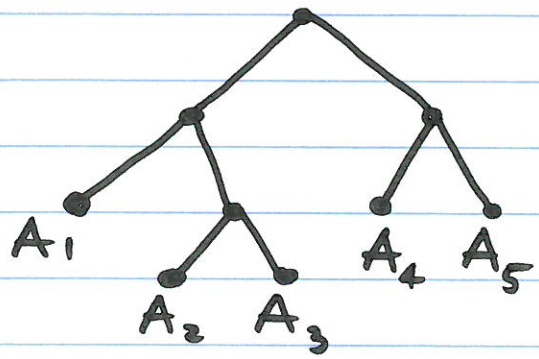
Polygon triangulation & Chained Matrix Prod.

Recall: Evaluation order of matrix chain $\leftrightarrow$ Parenthesis structure

$((A_1(A_2A_3))(A_4A_5)) \leftrightarrow$

$(((\ \ ))(\ ))$

well-formed parenthesis string

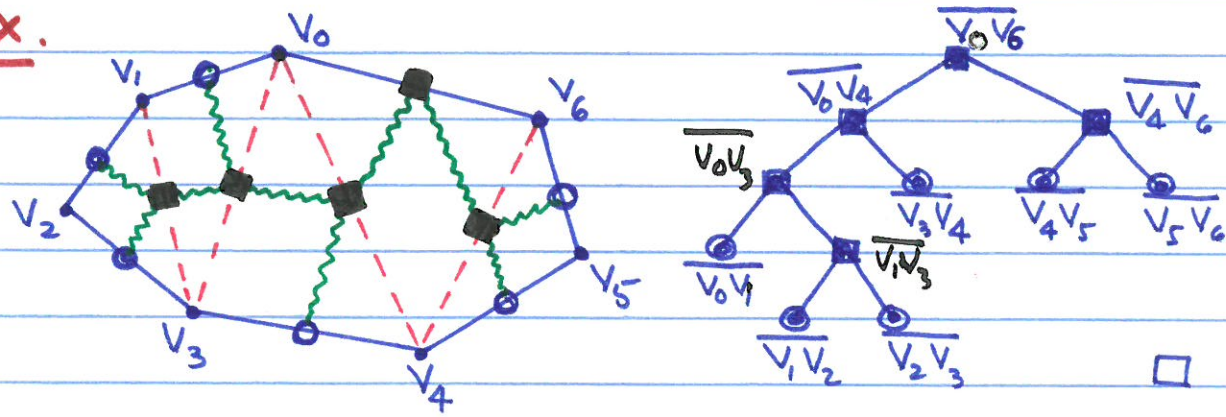


parse tree

Correspondence between triangulations and parse trees: A triangulation T of a polygon $P = \langle v_0, v_1, \ldots, v_{n-1} \rangle$ def. a parse tree as follows.

- every side corresp. to a leaf
- every chord " " an internal node
- node assoc. with $\overline{v_{n-1} v_0}$ is root

Ex.



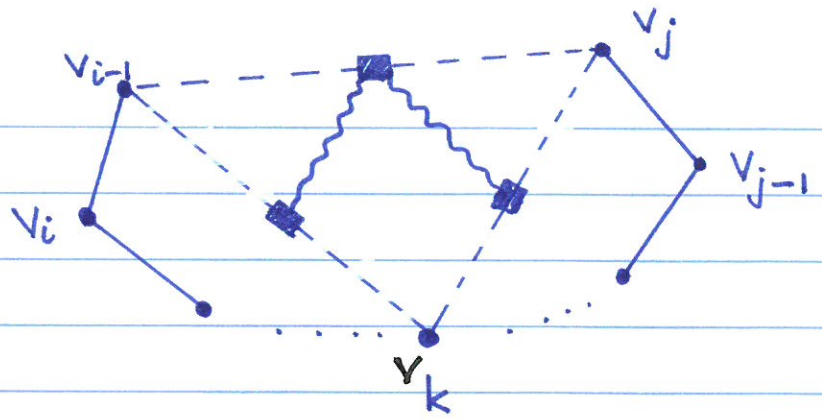
□

- Every subtree corresponds to a triangulation of a subpolygon.

Ex. Subtree rooted at $\overline{v_0 v_4}$ corresp. to triangulation of $\langle v_0, v_1, \ldots, v_4 \rangle$.

For $0 \leq i \leq j \leq n-1$:

m_{ij} := weight of opt. triangulation of $\langle v_{i-1}, \ldots, v_j \rangle$



Thus we have the recursion: $1 \leq i \leq j \leq n-1$:

$$m_{ij} = \min_{i \leq k \leq j-1} \left(m_{ik} + m_{k+1,j} + \omega(\Delta v_{i-1} v_k v_j) \right)$$

where $m_{ii} = 0$

```
POLYGON_TRIANGULATION (ω, n, v_0, ... v_{n-1})
    for i = 1 to n do m[i,i] = 0
    for l = 2 to n do
        for i = 1 to n-l+1 do
            j = i + l - 1
            m[i,j] = ∞
            for k = i to j-1 do
                q = m[i,k] + m[k+1,j] + ω(Δ v_{i-1} v_k v_j)
                if q < m[i,j] then
                    m[i,j] = q ; s[i,j] = k
    return m, s
```

record k when opt value found

3.3. Greedy Algorithms.

Example. Activity-selection Problem.

Given a set of n activities $S = \{a_1, a_2, \ldots, a_n\}$

Each activity has start time s_i

finish time $f_i > s_i$

If selected, a_i takes place during $[s_i, f_i)$

a_i and a_j are compatible if intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap, i.e., $s_i \geq f_j$ or $s_j \geq f_i$.

Goal. Find largest set of mutually compat. activities!

W.l.o.g: $f_1 \leq f_2 \leq \cdots \leq f_{n-1} \leq f_n$.

Optimal Substructure of Activity-Selection.

S_{ij} = set of activities a_ℓ s.t. $f_i \leq s_\ell$ and $f_\ell \leq s_j$

(i.e., a_ℓ can take place after a_i and before a_j)

$c[i,j]$:= size of opt. solution for S_{ij}

Then:

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \phi \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{otherwise} \end{cases}$$

Making Greedy Choice

Let $S_k := \{ a_i \in S \mid s_i \geq f_k \}$, $0 \leq k \leq n$,
= set of activities starting after a_k finishes

(Note: $S_0 = S$)

Claim. Let $S_k \neq \phi$. If $a_m \in S_k$ has the earliest finish time, then a_m is included in some opt. solution of S_k

Pf. Let A_k be an opt. sol. of S_k.
Let a_j be activity in A_k with earliest finish time.

Case (1): $a_j = a_m$ ✓

Case (2): $a_j \neq a_m$. Let $A_k' = A_k - \{a_j\} \cup \{a_m\}$

Clearly, A_k' is an opt. sol. of S_k and A_k' includes a_m. □

Strategy: Choose a_1 (w/ earliest finish time) for S_0 and proceed recursively!

ACTIVITY-SELECTOR (s, f, k, n) (for S_k)

```
m = k+1
while m ≤ n and s[m] < f[k]
    m = m+1
/* This finds first activity in S_k
   that starts after a_k */
if m ≤ n then
    return {a_m} ∪ ACTIVITY-SELECTOR(s,f,m,n)
else return φ
```

Note: ACTIVITY-SELECTOR (s, f, 0, n) solves the original problem.

Complexity: $\Theta(n)$

Iterative version of ACTIVITY-SELECTOR:

GREEDY-ACTIVITY-SELECTOR (s, f, n)

```
A = {a_1} ; k = 1
for m = 2 to n do
    if s[m] ≥ f[k]
        A = A ∪ {a_m}
        k = m
return A
```

Principle of greedy strategy:

- Cast problem s.t: greedy choice + 1 subproblem to solve
- Show there is an opt. sol. that includes greedy choice
- Show opt. substructure : opt. sol to subprobl. + greedy choice $\Longrightarrow$ opt. sol. to orig. probl.

Huffman Code

$C = \{c_1, \ldots, c_n\}$ be an alphabet s.t. each c_i is assigned prob. $f(c_i)$, $\sum_{i=1}^{n} f(c_i) = 1$

Encode each $c \in C$ by binary codeword $w_c \in \{0,1\}^+$ s.t.

- Decoding can be done unambiguously
- $\sum_{c \in C} f(c) \cdot |w_c|$ is minimized.

Prefix Codes. A set of binary codewords is a prefix code if no codeword is prefix of another.

Prefix Codes & Binary trees.

A prefix code can be associated with a bin. tree:

- Left (right) edge is labeled 0 (1)
- A codeword $\leftrightarrow$ string obtained by following a path from root to leaf

Note: The binary tree of an opt. prefix code is fully binary: every internal node has exactly 2 children.

By associating with leaves the prob. $f(c)$, $\sum_{c \in C} f(c) \cdot |w_c|$ = average depth of the tree.

Let W be a prefix code for C. Def.

$$H(W) := \sum_{c \in C} f(c) \cdot |w_c|$$

$H(W)$ is also the average number of bits to encode a character $c \in C$.

Input. A set C of characters and

$$f: C \to \mathbb{R}^+ \text{ s.t. } \sum_{c \in C} f(c) = 1$$

Output. A (opt.) prefix code W for C s.t. $H(W) = \sum_{c \in C} f(c) \cdot |w_c|$ is minimal.

Idea of Huffman's algor.

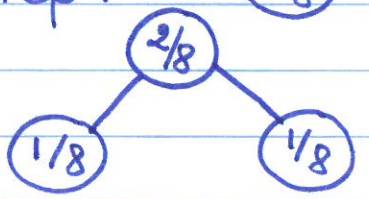
- A character c with low prob. occurs less often and is associated with longer codeword.
- Let c_{n-1}, c_n have lowest prob.
  - Construct prefix code W' for
    $$C' = \{c_1, \ldots, c_{n-2}, d_{n-1}\}$$
    where $f(d_{n-1}) = f(c_{n-1}) + f(c_n)$
  - If $W' = \{w_1, \ldots, w_{n-1}\}$
    then $W = \{w_1, \ldots, w_{n-2}, w_{n-1}0, w_{n-1}1\}$

Ex. $C = \{c_1, \ldots, c_5\}$ with prob. $\frac{1}{8}, \frac{1}{8}, \frac{1}{7}, \frac{1}{4}, \frac{10}{28}$

Constr. bin. tree as follows:

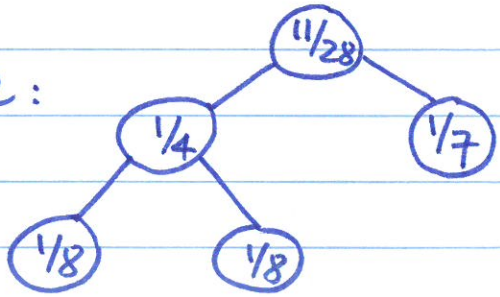
Initial step: 1/8 1/8 1/7 1/4 10/28

Step 1:



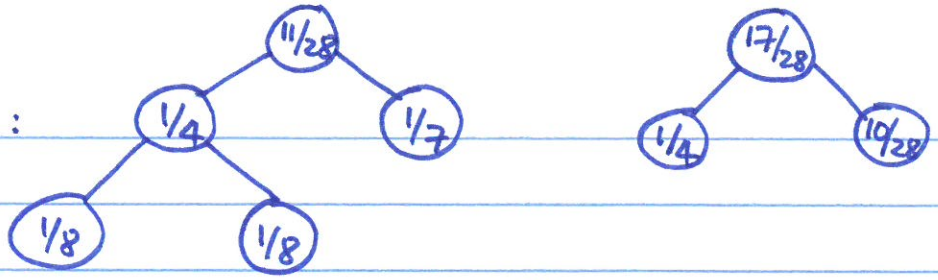
1/7 1/4 10/28

Step 2:

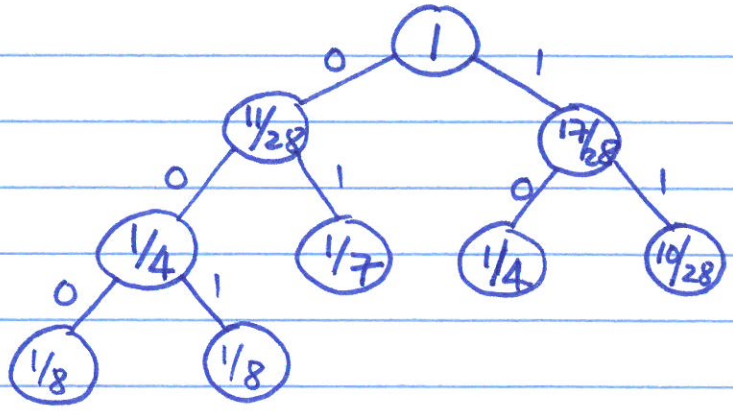


1/4 10/28

Step 3:



Step 4:



The opt. prefix code is {000, 001, 01, 10, 11} □

```
HUFFMAN(C, f)
  Q = C
  for i = 1 to n-1 do
    Let z be a new node
    Let x, y be roots of 2 trees in Q
        with least probabilities
    Let z be parent of x, y
    f(z) = f(x) + f(y)
```

Prop. If $x, y \in C$ have least prob., then there is an opt. prefix code s.t. $w_x = w0$ and $w_y = w1$ for some w.

Pf. Let T be a tree obtained from an opt. prefix code for C. Let a, b be sibling leaves at max. depth in T. Then a, b can be replaced by x, y to yield a tree T' whose aver. depth $\leq$ aver. depth of T ⊠

Prop. Let $x, y \in C$ have least prob. Let $C' = C - \{x, y\} \cup \{z\}$, where $f(z) := f(x) + f(y)$. Let T' be a tree repres. an opt. prefix code for C'. Then T obtained from T' by making x, y children of z is an opt. prefix code for C. ⊠

Complexity of HUFFMAN.

- Maintain a min heap for f
- Need to do insertion/deletion of minimal elements

$\Rightarrow$ Complexity is $O(n \lg n)$

3.4. Matroids & Greedy Methods

Def. Let S be a finite set and $\mathcal{I}$ be a nonempty family of subsets of S.

- $(S, \mathcal{I})$ is an *independent system* if

$$A \subseteq B \wedge B \in \mathcal{I} \Rightarrow A \in \mathcal{I}$$

i.e., it is *hereditary*

- Each element $A \in \mathcal{I}$ is called an *independent* set
- An independent system $(S, \mathcal{I})$ is called a *matroid* if it satisfies the following

$$A, B \in \mathcal{I} \wedge |A| > |B| \Rightarrow \exists x \in A - B : B \cup \{x\} \in \mathcal{I}$$

called the *exchange property*

Ex. Matric Matroid.

Let M be a matrix, S be set of rows of M. Let $\mathcal{I}$ be collection of all lin. independent subsets of S. Then $(S, \mathcal{I})$ is a matroid. ☒

Ex. Graphic Matroid

Let $G=(V,E)$ be undir. graph. Let $S=E$ and $\mathcal{I}$ = collection of sets of edges that induce a forest in G.

Claim. $(S,\mathcal{I})$ is a matroid.

Pf. (1) $(S,\mathcal{I})$ is obviously an indep. syst.

(2) $(S,\mathcal{I})$ sat. exchange property:

Let $A,B \in \mathcal{I}$ s.t. $|A|>|B|$ and consider forests F_A, F_B def. by A, B.

Fact. A forest F in G has $|V|-|F|$ trees.

Thus, F_A has fewer trees than F_B

$\Rightarrow$ Some tree in F_A have vertices in two different trees in F_B

$\Rightarrow \exists$ edge $e \in A$ connecting these 2 trees in F_B

$\Rightarrow$ Adding e to B doen't create cycle $\square$

Def. Let $(S,\mathcal{I})$ be a matroid and $A \in \mathcal{I}$. $x \notin A$ is called an extension of A if $A \cup \{x\} \in \mathcal{I}$

Ex. Consider graphic matroid. edge e is an extension of $A \in \mathcal{I}$ iff $A \cup \{e\}$ does not create a cycle.

Def. $A \in \mathcal{I}$ is maximal if it has no extension

Prop. All maximal indep. subsets in a matroid have the same size

Pf. Suppose otherwise that $A, B \in \mathcal{I}$ are maximal and $|A| > |B|$. Exchange prop. $\Rightarrow \exists x \in A - B : B \cup \{x\} \in \mathcal{I}$, contradicting the maximality of B □

Ex. In a graphic matroid, $A \in \mathcal{I}$ is maximal iff F_A is a spanning tree □

Def. A matroid $M = (S, \mathcal{I})$ is weighted if it is associated with a weight fct

$$w : S \longrightarrow \mathbb{R}^+$$

s.t. $w(x)$ is strictly pos. $\forall x \in S$.

w is extended to subsets $A \subseteq S$ by:

$$w(A) = \sum_{x \in A} w(x)$$

Question. How to compute max.-weight indep. subset A in a matroid?

Greedy algor. on weighted matroids

Ex. (Computing minimum-spanning trees)

Let $G=(V,E)$ be a connected graph with edge weight $c: E \to \mathbb{R}^+$. Then Computing min.-spanning tree is equiv. to finding max.-weight indep. subset in graphic matroid M_G with weight

$$c^*(e) = c_{max} - c_e$$

where $c_{max} = \sum_{e \in E} c(e)$ □

(Note: max.-weight indep. subset is maximal.)

Computing max.-weight ind. subset in a matroid

```
GREEDY((S,I), w)
    sort S in monoton. decreasing order
        w(x_1) ≥ w(x_2) ≥ ... ≥ w(x_n)
    A = ∅
    for i = 1 to n do
        if A ∪ {x_i} ∈ I
            then A = A ∪ {x_i}
    return A
```

We first show greedy-choice property:

Lem. Let x be first on list s.t. $\{x\} \in \mathcal{I}$. Then there is an opt. sol. A containing x.

Pf. Suppose otherwise there is an opt. sol. B s.t. $x \notin B$. Using exchange prop. constr. from x a set A by adding element of B to A. Since $w(x) \geq w(y)$ $\forall y \in B$, it follows that $w(A) \geq w(B)$. Thus, A is an opt. sol. containing x. ∎

Fact. (1) If x is extension of A, then x is extension of $\emptyset$

(2) If x is not an extension of $\emptyset$, then x is not an extension of any $A \in \mathcal{I}$.

Lem. (Optimal substructure property)

Let x be first element chosen by GREEDY.

Def. matroid $M' = (S', \mathcal{I}')$ by:

$$S' = \{ y \in S \mid \{y, x\} \in \mathcal{I} \}$$

$$\mathcal{I}' = \{ B \subseteq S' \mid B \cup \{x\} \in \mathcal{I} \}$$

Then finding a max.-weight ind. subset containing x reduces to finding max.-weight ind. subset in M'

Theorem. GREEDY $((S, I), w)$ returns opt. sol. □

Ex. (Task-scheduling probl. as matroid.)

Input.
- A set $S = \{1, 2, .., n\}$ of n unit-time tasks
- A set $d_1, .., d_n$, $1 \leq d_i \leq n$, of integer deadlines s.t. task i has to finish by time d_i, $i = 1, .., n$.
- A set $w_1, ..., w_n$ of (nonnegative) penalties s.t. w_i is incurred if task i is not finished by time d_i, $i = 1, .., n$.

Output. A schedule for S that minimizes the total penalty incurred for missed deadlines.

Let $\mathcal{S}$ be a schedule.

Task i is *late* in $\mathcal{S}$ if it finishes after d_i; otherwise it's *early*.

$\mathcal{S}$ is in *early-first* form if early tasks precede late tasks.

$\mathcal{S}$ is in *canonical* form if it's early-first and early tasks are scheduled in order of non-decreasing deadlines.

Fact. Every schedule $\mathcal{S}$ can be put into canonical form.

Def. A set A of tasks is *independent* if there exists a schedule s.t. no task in A is late.

For $t = 0, \ldots, n$, $N_t(A) := |\{ i \in A \mid d_i \leq t \}|$

Lem. A is independent $\iff N_t(A) \leq t$ for all $t = 0, \ldots, n$.

Pf. "$\Rightarrow$": Obvious

"$\Leftarrow$": If tasks in A are scheduled in order of non-decreasing deadlines, then no task is late. $\square$

Theorem. Let S be a set of unit-time tasks and $\mathcal{I}$ be the collection of all indep. sets of tasks. Then $(S, \mathcal{I})$ is a matroid.

Pf. (1) Hereditary: trivial.

(2) Exchange property: Consider 2 indep. sets of tasks A, B with $|A| < |B|$.

Let k be largest t s.t. $N_t(A) \geq N_t(B)$

Then $k < n$ and for $t = k+1, \ldots, n$:

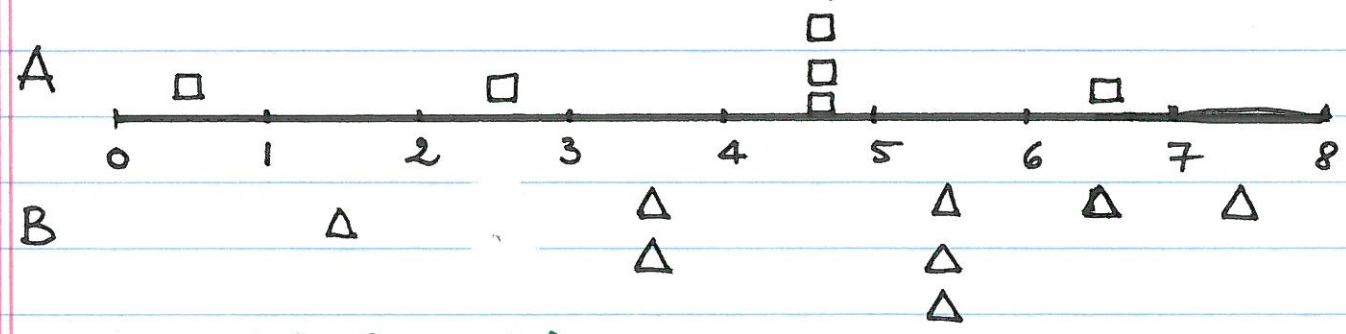
$$N_t(A) < N_t(B).$$

Choose $x \in \{ i \in B - A \mid d_i = k+1 \}$

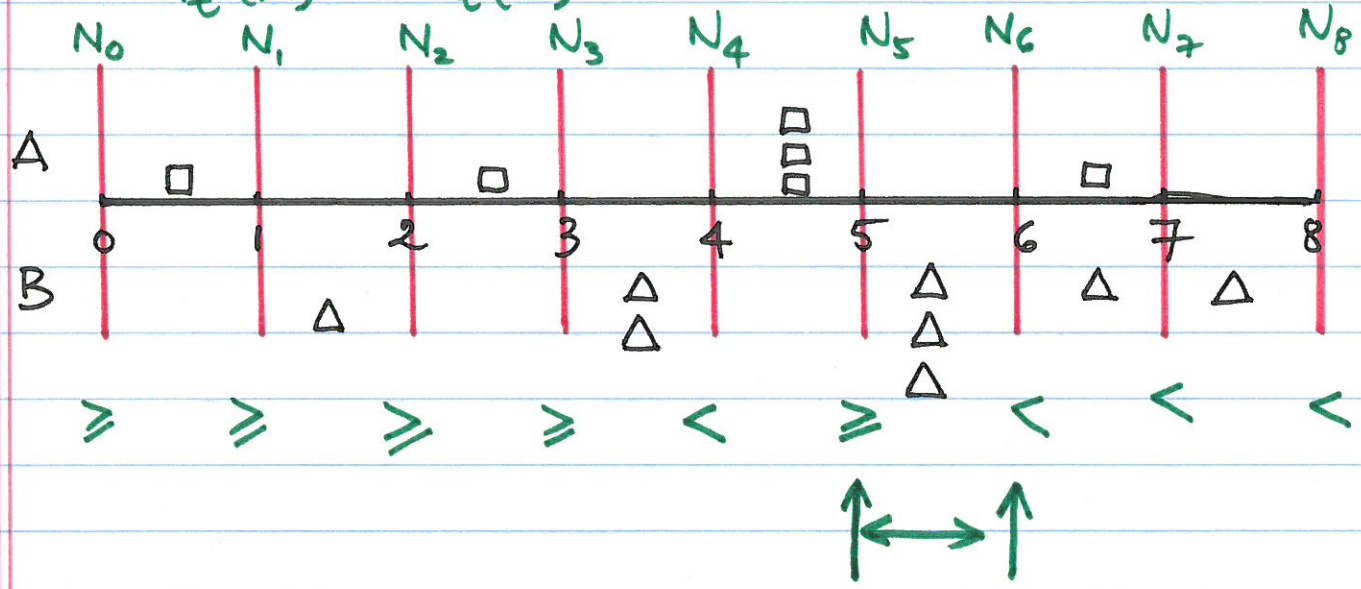
$\Rightarrow d_x = k+1$

$n = 8$ $\quad |A| = 6$ $\quad |B| = 8$

A and B are independent



$N_t(A)$? $N_t(B)$



Then $N_t(A \cup \{x\}) = N_t(A)$ for $t = 0, \ldots, k$ and $N_t(A \cup \{x\}) \leq N_t(A)+1 \leq N_t(B) \leq t$ for $t = k+1, \ldots, n$.

Thus, by Lemma above, $A \cup \{x\}$ is indep. ☒

We solve the task-scheduling probl.:

- Use greedy algor. to find max.-weight independent set A of tasks.
- Create an opt. schedule by first scheduling tasks in A in order of non-decreasing deadlines.

Complexity:

- Sort S in monot. decreasing order of penalties: $O(n \log n)$
- for loop: $O(n)$ steps
- checking for independence: $O(n)$ (using Lemma)

} $O(n^2)$

$\Rightarrow$ Total running time: $O(n^2)$