Activity - Selection

30 March 2025 01:35

Searches for locally optimal solution hoping that it will lead to a globally optimal one.

Min. spanning tree
Dijkstra's algorithm

i) Activity selection problem
ii) Basic Approach
iii) Applications → data compression (Huffman)
iv) Combinatorial structure → Matroids.
v) Application of matroids → unit time task (deadline & penalties)

Activity – selection problem

select the max. size set of mutually compatible activities (from a set of competing activities).

* $S = \{a_1, a_2 \dots a_n\}$ → for using the resource
* start time s_i and finish time f_i.

$$0 \leq s_i < f_i < \infty$$

* 2 activities can't overlap $[s_i, f_i)$ and $[s_j, f_j)$ only if $s_j \geq f_i$. { ∵ f_i not included hence }

assumption : → activities → sorted

$$f_1 < f_2 < f_3 < f_4 \dots < f_{n-1} < f_n.$$

Let the activities be a_i where $1 \leq i \leq n$ and $I \in Z$

I	1	2	3	4	5	6	7	8	9	10	11
si	1	3	0	5	3	5	6	8	8	2	12
fi	4	5	6	7	9	9	10	11	12	14	16

$\{a_3, a_9, a_{11}\}$ → compatible.
$\{a_1, a_4, a_8, a_{11}\}$ → largest compatible.
$\{a_2, a_4, a_9, a_{11}\}$ → another largest.

D.P APPROACH

$a_{i'} < s_{i,j} < a_j$ $\{ s_{ij}$ → set of activities in b/w a_i & a_j $\}$.

$a_i < s_{ij} < a_j$ $\quad \{ S_{ij} \rightarrow$ set of activities in b/w a_i & $a_j \}$

a_i → finishes ; s_{ij} → starts

A_{ij} → max. set which also has some activity a_k

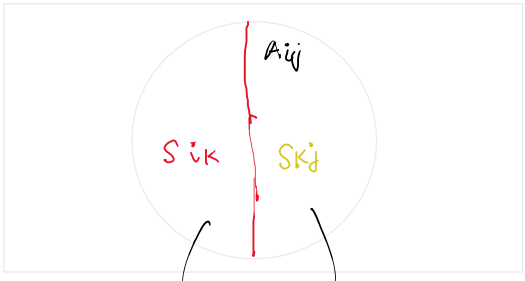
$A_{ij} \equiv a_i < s_{ij} < a_j$ (s_{ij} has some a_k)

$$S_{ij} = S_{ik} + S_{kj}$$

mutually compatible act. b/w $i \rightarrow j$ (via k)

$A_{ik} = A_{ij} \cap S_{ik}$

$A_{kj} = A_{ij} \cap S_{kj}$



$A_{ij} \cap S_{ik}$ (all set end time $< k$) $\qquad A_{ij} \cap S_{kj}$ (all set with start time $> a_k$ end time $< a_j$)

$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$

Max size set of A_{ij}

is $S_{ij} = |a_{ik}| + |a_{kj}| + 1$

Optimal soln → A_{ij} → S_{ik}, S_{kj} } subprob.

Size of optimal solution in $S_{ij} = c[i,j]$

then

$$c[i,j] = c[i,k] + c[k,j] + 1$$

∴ The Recursive call is :→

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

GREEDY APPROACH

choose an activity to add to our sol^{n} → greedy choice.

* THE greedy choice :→

i.) should leave resources for more activities in future.

choose one with the first finish time.

→ if activities → monotonic finish time → choose a_1.

→ only one subproblem → activities that start after a_1 finishes.

I	1	2	3	4	5	6	7	8	9	10	11
si	1	3	0	5	3	5	6	8	8	2	12
fi	4	5	6	7	9	9	10	11	12	14	16

select activities that start after a_1 finishes i.e. after 4

Do now, $S_K = \{a_i \in S : s_i \geq f_k\}$ → set of all starting after 4.

?? Is that an optimal solution for the subproblems also?

Pg:→ 418 Th 16.1

$S_K \neq \varphi$, $a_m \in S_K$ and a_m → earliest finish time

then $a_m \in A_K$ → max. size subset of mut. compatible activities

∴ The solution is always choose first finishing time → make subproblem

Repeat till no act. left.

GREEDY → greedy choice → solve subproblem (TOP-DOWN)

D.P → solve subproblem → solve problem (BOTTOM-UP)

RECURSIVE GREEDY ALGORITHM

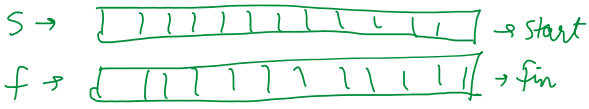
I	1	2	3	4	5	6	7	8	9	10	11
si	1	3	0	5	3	5	6	8	8	2	12
fi	4	5	6	7	9	9	10	11	12	14	16

s → start K → index of the subproblem S_k } Returns A_K (max. size of mutually comp activities)
f → finish n → size of original problem.

sort (acc. to finish time → O(nlogn)).

a_0
$f_0 = 0$ } fictious to start.

S_0 → entire set of activities of S.



RECURSIVE - ACTIVITY - SELECTOR (s, f, 0, n) → func^n call

RECURSIVE - ACTIVITY - SELECTOR (s, f, k, n)

1. m = k+1
2.) while m ≤ n and s[m] < f[k]
3.) m = m+1 ;
4.) if m ≤ n
5 return {am} ∪ RECURSIVE - ACTIVITY - SELECTOR (s, f, m, n)
6. else return φ

I	1	2	3	4	5	6	7	8	9	10	11
si	1	3	0	5	3	5	6	8	8	2	12
fi	4	5	6	7	9	9	10	11	12	14	16

m = 1
s[1] = 1 ≮ f[0]

{a1} ∪ (s, f, 1, n)

m = 2
s[2] = 3
f[1] = 4
m = 3
s[3] = 0
f[1] = 4
m = 4
s[4] = 5
s[1] = 4 (X)

{a1, a4} ∪ (s, f, 4, n)

// 2 & 3 → if s[m] < f[k]
we are not including it
and we move on.
if greater we add it to the set.

Similarly the rest, Recursive - Activity selector → O(n)

ITERATIVE APPROACH

GREED - ACTIVITY - SELECTOR (s, f)

1.) m = s.length();
2.) A = {a1}.
3.) k = 1
4.) for i = 2 to m.
if (s[i] ≥ f[k])

4.) for $i = 2$ to m.

5.) if $(s[i] > f[k])$

6.) $A = A \cup \{a_i\}$

7.) $k = i$

8.) return A.

→ T.C = $O(n)$ { assuming activities already sorted }.

Elements of Greedy Strategy

02 April 2025 00:13

What we did.

i) determined optimal substructure

ii) Recursive (D.P) approach

iii) Making greedy choice, one substructure

iv) Prove safe to make greedy choice.

v) Recursive greedy

vi) Iterative greedy

GREEDY CHOICE

i) Best for current problem, then for subproblems. (Unlike D.P)

ii) greedy choice should be optimal for sub problems also.

OPTIMAL SUBSTRUCTURE

↳ OPTIMAL SOL^{n} to problem → optimal sol^{n} to subproblem.

Fractional Knapsack Problem

03 April 2025 00:42

The problem is same like the 0-1 knapsack problem except the fact that now can fill fractions of items into the knapsack

Consider items like rice, flour, wheat, bajra etc.

greedy choice :→ take out the value per unit i.e $\frac{v_i}{w_i}$

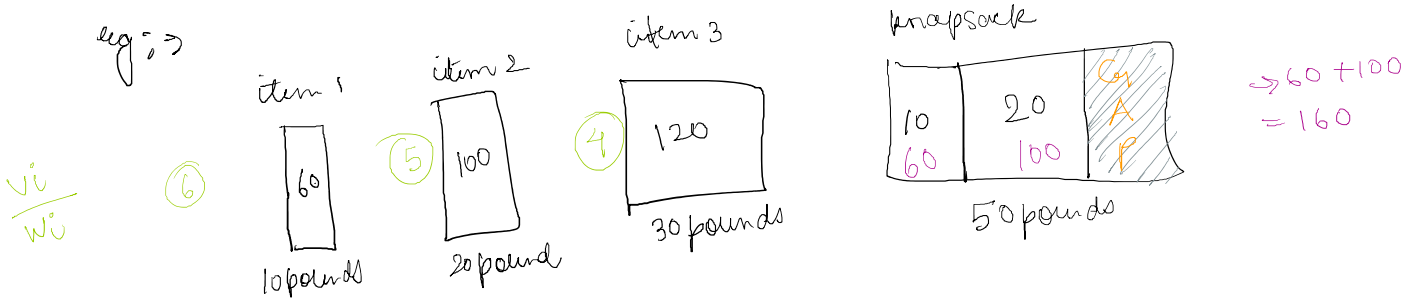
→ fill the knapsack until the item is over → proceed to next item.
↳ or knapsack is full.

Once we have the value per unit, sort it in order to get the greedy solution in $O(n \log n)$ time.

It also has the greedy choice property of optimal substructure.

greedy does not work in 0-1 knapsack → since we have gaps left in 0-1 knapsack if greedy applied.

eg :→



Coin problem

03 April 2025 01:01

$\{1, 2, 5, 10, 20, 50, 100, 200\}$.

Find Min no of coin to make a total of n.

eg:→ $n = 520$

MIN = 200 + 200 + 100 + 20.

GREEDY APPROACH

choose max possible coin.

Reason:→ Any 2 coins of same denomination can be replaced.

eg:→ 1+1 = 2 50+50=100 ∴ {1, 5, 10, 20, 100} can appear at most once.

5+5=10 100+100=200

10+10=20.

Reason:→ 3 coins of 2+2+1 can be combined to have 1 coin 5.

20+20+10 ⟶ 50.

Hence not possible to construct n, optimally by using coins smaller than n.

FAILS in general coin problem.

Eg:→ $\{1, 3, 4\}$

for $n=6$, → $\{4, 1, 1\}$

but optimal = $\{3, 3\}$.