# Final Exam

G. krishna Sree
12222?1765

Question 9 :

Part-A :

Given knapsack-II problem,

OPT(i,v) = min weight subset of items, 1...i which results value exactly "equal to" V

$$OPT(i,v) = \begin{cases} 0 & \text{if } v=0 \\ \infty & \text{if } i=0, v>0 \\ OPT(i-1,v) & \text{if } v_i > V \\ \min\{OPT(i-1,v),\ w_i + OPT(i-1, v-v_i)\} & \text{otherwise} \end{cases}$$

To prove, knapsack-II is NP complete as discussed in lecture,

step 1 : show knapsack-II is in NP

step-2 : choose an NP complete, here given subset-sum problem.

step-3 : Prove that subset sum $\leq_p$ knapsack

Let's show step-1 :

Proof:- Run-time complexity of knapsack II is $O(nV^*)$ : $O(n^2 V_{max})$ as $(V^* \leq nV_{max})$

$\Rightarrow$ $V^*$ is the optimal value and not polynomial in input size

$\Rightarrow$ It cannot run in polynomial time, if $V^*$ is a very large value (Ex. $V^* \approx 2^n$)

→ Moreover, knapsack-II can be certified in polynomial time

i.e, when given set of items 1 ... i we can check their total value = V and their total weight is $\leq W$ (weight limit) in linear time.

Hence, knapsack II is in NP.

Now, for step-2 :

As given in question, subset-sum problem is NP complete

For final step, lets prove subset sum can be reduced to knapsack problem in polynomial time i.e subset sum $\leq_p$ knapsack

Proof:- As we already know that all problems in NP can be reduced to subset sum problem, so it can also be reduced to knapsack problem.

$\Rightarrow$ Create a knapsack instance

Given instance $(u_1, \ldots u_n, U)$ of subset-sum

$$V_i = W_i = u_i \qquad \underbrace{\sum_{i \in S} V_i \geq V \iff \sum_{i \in S} u_i \geq U}_{\text{left part}} \qquad \underbrace{\iff \sum_{i \in S} u_i = U}_{\text{right part}}$$

$$V = W = U \qquad \sum_{i \in S} w_i \leq W \iff \sum_{i \in S} u_i \leq U$$

If left part condition is satisfied i.e. we get subset of items whose sum is exactly equal to $V = U$ (as we select items whose total value $= V$, so i.e. we are indirectly selecting integers whose sum is equal to $U$) then right part condition is also satisfied.

i.e eventhough total weight of selected items in knapsack is $\leq W$

$\because$ $W_i = V_i$ we get total weight $= W$

$\therefore$ By getting solution to knapsack, we are able to get solution for subset_sum problem. Similarly, if no solution for knapsack the we get no for subset_sum problem.

$\therefore$ subset sum $\leq_p$ Knapsack. II

$\therefore$ from concluding 3 steps, we can say that knapsack II is NP-complete problem (as discussed in lectures)

$\Rightarrow$ As, steps taken to reduce subset-sum to knapsack II are in polynomial time. Here, we have used "special case to general case" reduction strategy as we showed that special case of general knapsack problem is subset-sum problem.

Part - B :-

As we know by definition of knapsack II algorithm - optimisation version,

the solution is subset of items whose total weight $\leq W$
total value $= V$

$\rightarrow$ so, it is not a decision problem

Since decision problem is defined as yes or NO

$\therefore$ the optimized knapsack-II problem is not a decision problem.

so, it is not in NP, $\therefore$ Not in NP-complete

because NP consists only decision problems.

Part-C:

we have seen approximization algorithm for Knapsack II

As discussed in lecture, we got

solution by our algorithm $\geq \left(\frac{1}{1+\varepsilon}\right)$ optimal solution.

to get $1/2$- approximation algorithm,

we can choose $\varepsilon = 1$, then we get $1/2 \cdot$ (optimal solution)

on condition $\left(\theta = \frac{V_{max}}{n}\right)$

$\Rightarrow$ this way we can achieve $1/2$- approximization

# Cheat sheet

G. Krishna Sree
12222717G5

## classes P and NP

P: Decision problems for which there is a poly-time algorithm.
NP: Certification algorithm intuition.
→ decision problems for which there exists a poly-time certifier.

3-SAT, HAM-CYCLE - NP

$P \subseteq NP \subseteq EXP$

$P \,!= NP$

$X \le_p Y$ - X can be reduced to Y NP in polynomial time.
If Y can be solved in P time, X can also be solved.

Design Algo:
Establish intractability:- If $X \le_p Y$ & X cannot be solved in poly time, then Y cannot be solved in poly time.
Establish equivalence:- If $X \le_p Y$ & $Y \le_p X \Rightarrow X \equiv_p Y$

## Reduction strategies:

1) Reduction by simple equivalence.
Independent set $\equiv_p$ vertex-cover
Given G, show S - independent set of size k
V-S - vertex cover of size $k' = n-k$

2) Reduction from special case to general case.
→ vertex-cover $\le_p$ set-cover

3) Reduction by encoding with gadgets.
3-sat $\le_p$ Independent-set

Transitivity:- If $X \le_p Y$ & $Y \le_p Z$ then $X \le_p Z$

So:
3-SAT $\le_p$ Independ. set $\le_p$ Vertex cover $\le_p$ set cover

## NP completeness:

A problem in Y in NP with the property that
$\forall$ problem X in NP, $X \le_p Y$

Prove NP completeness of problem Y:-
Step 1: Show that Y is in NP
Step 2: Choose an NP-complete problem X
Step 3: Prove that $X \le_p Y$

Justification:- If X is an NP-complete problem, & Y is a problem in NP with the property that $X \le_p Y$, then ~~is com~~ Y is NP complete

Pf:- W be in NP, then $W \le_p X \le_p Y$
by transitivity $W \le_p Y$, ∴ Y - NP complete

## Approximation Algorithms:

PTAS:- Polynomial Time Approximation Scheme.
Approximation algor
→ $(1+\epsilon)$- approximation
- $\epsilon > 0$
→ via rounding and scaling.

Knap-sack - NP complete.
Subset sum $\le_p$ Knap-sack

Knap. sack II

OPT(i,v) = min weight subset of items 1...i that yields value exactly v

case-1: OPT - x i
✓ — 1...i-1 for value = v

case 2: OPT - ✓ i
✓ — 1...i-1 for value = $v - v_i$

$$OPT(i,v) = \begin{cases} 0 & v=0 \\ \infty & i=0, v>0 \\ OPT(i-1,v) & v_i > v \\ \min\{OPT(i-1,v),\ w_i + OPT(i-1, v-v_i)\} & \text{other} \end{cases}$$

$O(nv^*) = O(n^2 V_{max})$
↑ not polynomial in input size.

$v^* \le n V_{max}$

1) Stable Matching for Roommate Problem - doesn't exist.

Propose - and - Reject - Algo: Stable matching $\not\rightleftarrows$ Perfect matching

POC:- O1: Men propose to women in ↓ order.

O2: Women once matched, she only trades up.

GS algo guarantee to find stable matching for any instance.

GS - man optimal assignment - yields one stable matching - $O(n^2)$

2) Interval Scheduling: Earliest Job first

consider the jobs in increasing order of finish time.

Implementation - $O(n\log n)$

3) Interval Partitioning:- find min no: of classrooms

No: of classrooms needed $\geq$ depth.

optimal schedule = depth.

Greedy algo:- Consider lectures in ↑ order of start time.

Implementation - $O(n\log n)$ - using priority queue.

4) Scheduling to Minimizing Maximum lateness:-

lateness = $\max\{0, t_j - d_j\}$

Greedy algo - Earliest deadline first

has no inversions ← optimal with no ideal time

5) optimal offline caching:-

Greedy algo: farthest - in - future.

Any unreduced schedule S, can transform into reduced schedule with no more cache misses

* LRU - k competitive.

Greedy - Analysis Strategies

1) Greedy algo stays ahead

2) Exchange argument

- transform any solution to greedy without hurting its quality

3) Structural:

discover a simple structural bound. to show our algo achieves this bound.

Shortest Path Problem:-

Binary heap - $O(m\log n)$

Fib heap - $m + n\log n$

$\pi(v) = \min_{e=(u,v)}(d(u) + l_e)$

MST:- cayley's theorem: $n^{n-2}$ spanning trees

sum of edge weight is minimized

Eg: Network design

Kruskal:- Start with $T = \phi$, consider edges in ascending order of cost.

cutsets: cut - set of nodes 'S'

cutset - edges with one end point in 'S'

uses cycle property

Prim's:-

start with some root node s, grows a tree T.

each step add min edge e to T which has exactly 1 endpoint in T

uses cut-property.

use priority queue for $O(\log n)$

Amortize Cost

Aggregate Method:- T(n) - worst case running.

amortized cost = $T(n)/n$

Accounting:-

Ex: a: 1 if i=1 i.e $i = 2^0$; 2 $i = 2^k$; 3 otherwise

c: 1 if i=1; i $i = 2^k$; 1 otherwise

2 extra credit is saved for later use. Credits saved before $i^{th}$ operation, $2(2^k - 2^{k-1} - 1)$

Potential:

$\phi(D_0) = 0 \quad \phi(D_i) = 2i - 2^p \quad 2^p \leq i < 2^{p+1}$

$a = c_i + \phi(D_i) - \phi(D_{i-1})$

$i = 2^k \quad = i + \{2i - 2^{k+1} - 2(i-1) - 2^k\} = 2 = O(1)$

$i \neq 2^k \quad \leq 1 + \{2i - 2^p - 2(i-1) - 2^p\} = O(1)$

Splay trees

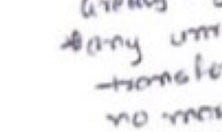
1) depth d(u) = root to v

2) height = v to leaf

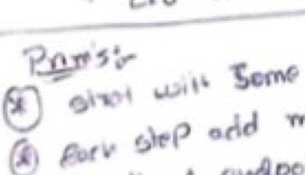
3) size |u| = # nodes under v

4) Min height = $O(\log n)$

Roller Coasters:- amortized cost of search, insert, delete = $O(\log n)$



Zigzag:



CHEAT SHEET

KRISHNA SREE

12222 91765