Algorithm Analysis and Design - Assignment 3

Aryaman Manish kolhe
2022121002

9th Nov. 2022

1. $G = (V, E)$ $\quad n = |V|$

G is undirected.

Weight of node v_i is w_i

$\exists (v_i, v_j) \in E$ iff $|i-j| = 1$ [Given]

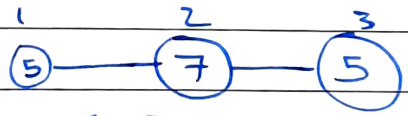
$i, j = 1, 2, \ldots, n$ in G. G is called a path

Problem -

Find the maximum weight independent set in such a path, G.

(a) Greedily picking the maximum weight node will not always work.

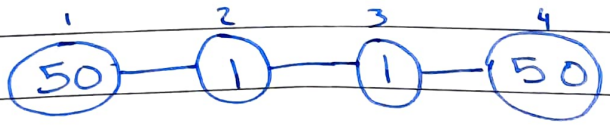
Simple Example : node 1 2 3



The returned output is $\{2\}$ with weight 7 but the correct answer is $\{1, 3\}$ with weight 10.

(b) This algo will not work because it is not cleverly skipping nodes.

Example : node 1 2 3 4



The returned output is $\{1, 3\}$ with weight 51, but the correct answer is $\{1, 4\}$ with weight 100.

(c) We can use the Dynamic Programming logic where the algo either considers the node, or does not consider it.

If the algo considers node i, then the final result will include W_i. From that point, node $i-1$ cannot be considered, since i was considered. So we'll go and explore from node $(i-2)$ onwards.

else
Or, explore from node $(i-1)$ onwards.

∴ let $OPT(i) = 0$ for $i < 1$
For $i \geq 1$,
When we consider node v_i,
$OPT(i) = W_i + OPT(i-2)$

When we do not consider v_i,
$OPT(i) = OPT(i-1)$

$$\therefore OPT(i) = \begin{cases} \max\{W_i + OPT(i-2), OPT(i-1)\} & i \geq 1 \\ 0 & i < 1 \end{cases}$$

To get the elements belonging to the maximum independent subset, we need to do some bookkeeping as $OPT(i)$ is calculated (store i as v_i is considered)

The answer = OPT(n),

and since OPT(i) takes can be done in O(i),
OPT(n) is O(n).
(takes constant time to get OPT(i) from
OPT(i-1) and opt(i-2))

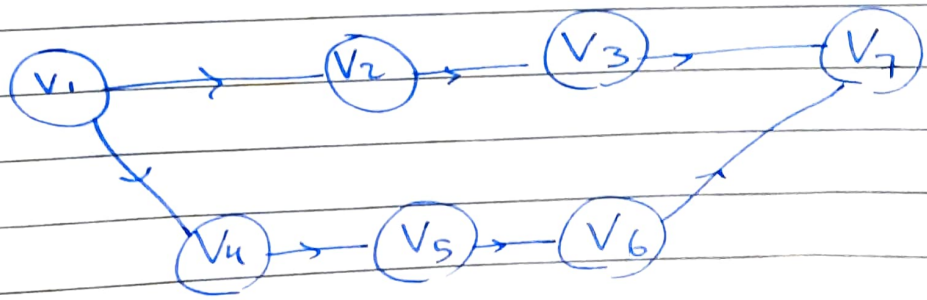
2. $G = (V, E)$ $V = \{v_1, \ldots, v_n\}$ directed

G is ordered if:
→ Each edge goes from lower to higher index node
→ $\forall v_i \neq v_n$, v_i has ≥ 1 edge leaving it.

longest path beginning at v_1 and ending at v_n.

(a) This approach does not work when choosing a higher index node is more profitable in the long run.

e.g.



The algorithm returns 3, as the path found is $v_1 \to v_2 \to v_3 \to v_7$,
but the correct answer is 4, with path $v_1 \to v_4 \to v_5 \to v_6 \to v_7$

(b) Similar approach to 1.

~~$f(i) = \max_j \{ j \mid j \in V \text{ and } j \to i \in E \}$~~

~~$f(i)$ gives the~~

We need to consider all edges that are incoming wrt node i.

→ If there's no path from v_1 to node i, then count the path length as $-\infty$.

→ Path from v_1 to v_1 has length 0.

∴ $OPT(1) = 0$

Algorithm -

Initialize DP as an array with n elements

DP [1 ... n]

DP [1] = 0

.

For i = 2 , to n :

$x = -\infty$

For all $(j, i) \in E$:

if DP[j] != $-\infty$ and $x < DP[j] + 1$

$x = DP[j] + 1$

DP [i] = x

Return DP[n]

We can find the path by keeping track of which node is considered as x is updated.

The way this works is that x keeps track of the longest path considering all incoming edges to node i.

The upper bound number of incoming edges is n, and the outer for loop runs n-1 times,

∴ Time Complexity = $O(n^2)$

3. n weeks
Supply s_i

Company A: fixed rate r per pound. ($r \cdot s_i$)

Company B: C per week for 4 weeks. (4C)

This can be solved with the same DP approach as before.

If company A is considered,
$OPT(i) = rs_i + OPT(i-1)$
↳ free to make a new choice.

If company B is considered
$OPT(i) = 4C + OPT(i-4)$
↳ contract of 4 weeks.

and the cost needs to be minimized.

$\therefore$ $OPT(i) = \min(\pi s_i + OPT(i-1), 4c + OPT(i-4))$

We can start with $i = 1$ and build upwards $\rightarrow$ $OPT(i) = 0$, $i \leq 0$.

To find the answer, compute $OPT(n)$.

To get the exact schedule, for each i, keep track of which company is chosen.

4. n servers.
$S_1, \ldots S_n$.

For servers from 1 to i, let the minimum cost be $OPT(i)$, given that a copy is placed at server i.

We want to search over all possible places to put the highest copy of the file before i, in the optimal position (let this be position x)

The cost for all servers upto x is $OPT(x)$;

The cost for servers $x+1, x+2, \ldots, i$ is the sum of all access costs $x+1$ to i.
$= 0 + 1 + \ldots + (i - x - 1) = {}^{i-1}C_2$.

Additionally, we give c_i to place the server at i.

To minimize the cost, we can use the following recurrence.

$$OPT(i) = c_i + \min_{0 \le x < i} \{ OPT(x) + \sum_{j=x+1}^{i-1} c_j \}$$

Initially, $OPT(0) = 0$ and $c_2 = 0$.

The values of OPT can be built up in order of increasing i in time $O(i)$ for iteration i;

Which gives a total running time of $O(n^2)$. [Polynomial].

$OPT(n)$ gives the solution. To find the configuration, we can ~~update~~ do some bookkeeping each time $OPT(i)$ is found (to trace back).

5 (a). The following algorithm checks the validity of the given $d(v)$'s in $O(m)$ time -

If for any edge $e = (v, w)$, we have $d(v) > d(w) + c_e$, then we can reject it.

(since if $d(w)$ is correct, then $\exists$ path $v \to t$ from w with cost $d(w) + c_e$).

Minimum distance from $v \to t$ is at most this value. so $d(v)$ must be incorrect.

Consider G' formed using G and removing all edges except those $e = (v, w)$
$\Rightarrow d(v) = d(w) + c_e$.
Then check if every node has a path to t in G' (in $O(m)$ time by reversing edges and using DFS).

If any node fails to have such a path, then reject it. Else accept it.

Note how if $d(v)$ was correct $\forall v \in V$ then if we consider those edges on the shortest path from any node v to t, these edges will all be in G'.

$\therefore$ if $\nexists$ path to t in G', then reject the node

$\therefore$ The algo rejects a set of distances if they are incorrect.

The algo also accepts the correct distances.

Proof -

Let $d'(v)$ = Actual distance in G' from $v \to t$.

It is enough to show that $d'(v) = d(v)$. $\forall v$.

For the path $v \to t$ in G',
real cost of path = $d(v)$.

There may be shorter paths, but we know that $d'(v) \leq d(v)$ $\forall v$.

Suppose $\exists v \Rightarrow d'(v) < d(v)$.

For the shortest path $P : v \to t$,
let x = last on $P \Rightarrow d'(x) < d(x)$.

Let y be the node in P, after x.

By definition of x, $d'(y) = d(y)$

Since P is the shortest path to t from v, it is also the shortest path from all nodes on P.

So $d'(x) - d'(y) < c_e \Rightarrow e = (x, y)$.

$\Rightarrow d(x) > d'(x) - d(y) + c_e$.

(contradiction)

Since algo should have rejected on inspecting e. $\therefore d'(v) = d(v)$. Q.E.D.

(b) Given distances to sink t, we can efficiently calculate distance to another sink t' in $O(m \log n)$ time. If all costs were ≥ 0, we can just use Dijkstra's algo.

otherwise, modify costs as follows.
for $e = (v, w)$, $c'_e = c_e - d(v) + d(w)$.
Note that $c'_e \geq 0$, since if $c'_e < 0$ then,
$d(v) < d(w) + c_e$, which is not
possible if d reflect true distances to t

Consider any path P $x \to t'$.
real cost of P = $c(P) = \sum_{e \in P} c_e$.

Modified cost of P = $c'(P) = \sum_{e \in P} c'_e$

$$= \sum_{e \in P} c_e - d(v) + d(w)$$

Note that all except for the first and last node in P occur once positively and once negatively
$\Rightarrow$ $c'(P) = c(P) - d(w) + d(t')$

∴ Modified cost $x \to t'$ differs from the real cost of the same path by $d(t') - d(x)$, which is constant.

∴ The set of minimum cost paths from x to t' under c' is the same as under c.
∴ since min distance $x \to t'$ under c' is available, we can calculate the min distance under c by adding $d(x) - d(t')$

Summary :
→ Calculate modified costs
→ Run Dijkstra from t' (by reversing edges), then adjust distances.

∴ Total time taken

$= O(m + m \log_n m) = O(m \log_n)$ //

References and Collaborators-

- Algorithm Design – Kleinberg and Tardos
- Ishan Kavathekar (discussed insights
- Aaryan Sharma in problems 4 and 5)