Daryl Fernandes 20200120049(72)

## Assignment 2

1) - Let $G = (V, E)$ be a weighted directed graph of a set V of vertices & a set of E edges. A node in a graph represents a city & an edge $\langle i, j \rangle \in E$ represents a path between two cities i and j. The cost (weight) of an edge $\langle i, j \rangle \in E$ is given as $C_0 > 0$ for all i & j and $j \in V$ & $C_s = \infty$ if an edge $\langle i, j \rangle \in E$

- Let $|V| = n$ & $n > 1$. A tour of G starts at any node $i \in V$ & ends at j by visiting all other nodes in $(V - (i))$ only once.

- Thus a tour of G is simple directed cycle including each vertex in V. The sum of all the costs of all paths (or edges) included in a tour determines the cost of a tour.

- The travelling salesman problem is to determine a tour with smallest cost value.

Solution by Dynamic Programming

- Consider $G = (V, E)$ be a given weighted directed graph where V is a set of nodes (cities) & E is a set of edges (paths).

Daryl Fernandes 20200120004 (72)

- Assume that a tour at a node 1 & ends at that same node. Each tour includes an edge $\langle 1, k \rangle$ for some $k \in V - (1)$ & a path from a node k to a node 1 passing through each node in $V-(1-k)$ only onee.

- By an optimal substructure property; if a tour of G is optimal, then the path from a node k to node 1 must be the optimal path from k to 1 passing through all nodes in $V - (1, k)$. Thus it obeys the principle of optimality.

- Let $g(i, S)$ be the cost of the shortest path from a node i to a node 1, passing through all nodes in a set S.
Applying the principle of optimality an optimality an optimal cost of a tour starting from a node 1 in G is given as:

$$g(1, V-(1)) = \min_{2 \le k \le n} (C_{1k} + g(k, V-(1,k)))$$

- Hence, the generalied recursive formula for $g(i, S)$ is given as:

$$g(i, S) = \min_{j \in S} (C_{ij} + g(j, S-(j)))$$

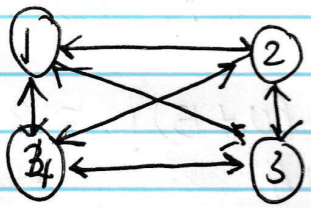
Initially, $g(i, \phi) = C_{i1} \quad 1 \le i \le n$.

Daryl Fernandes 20200120004(72)

- To compute $g(1, V-\{1\})$ firstly $g(i,S)$ for all $S$ with $|S| = 1$ is calculated, then $g(i,S)$ for all $S$ with $|S| = 2$ is calculated & so on.

- To trace the shortest cost tour, the value of $j \in S$ that produces the minimum cost $g(i,S)$ is recorded. Let $J(i,S)$ stores such value of $j$ associated with each $g(i,S)$. Finally, $J(1, V-\{1\})$ gives a path of the shortest tour of G starting from 1.

Example:




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 10 | 15 | 20 |
| 2 | 5 | 0 | 9 | 10 |
| 3 | 6 | 13 | 0 | 12 |
| 4 | 8 | 8 | 9 | 0 |

Solution:

1) $|S| = 0$

$g(2, \phi) = c = 5 \quad d = 1$

$g(3, \phi) = c_{31} = 6 \quad d = 1$

$g(4, \phi) = c_{41} = 8 \quad d = 1$

2) $|S| = 1$

$g(2, \{3\}) = \min\{c_{23} + g(3, \phi)\} = \min\{9+6\}$

$= 15$

$\therefore d = 3.$

Daryl Fernandes 20200120064 (72)

$g(2,\{4\}) = \min(C_{24} + g(4,\phi)\} = \min\{10+8\} = 18$
$d=4$

$g(3,\{2\}) = \min(C_{32} + g(2,\phi)\} = \min\{13+5\}$
$= 18 \quad d=4$

$g(3,\{2\}) = \min C C_{34} + g(4,\phi)\} = \min\{12+8\}$
$= 20 \quad d=2$

$g(4,\{2\}) = \min\{C_{42} + g(2,\phi)\} = \min\{8+5\}$
$= 13 \quad d=4$

$g(4,\{3\}) = \min\{C_{43} + g(3,\phi\} = \min(9+6)$
$= 15 \quad d=3$

③ $|S| = 2$

$g(2,\{3,4\}) = \min\{C_{23} + g(3,4)\}, C_{24} + g(4,\{3\}\}$

$= \min\{(9+20), (0+5)\} = 25$
$d(2,\{3,4\}2 = 4$

$g(3,\{2,4\}) = \min\{C_{32} + g(2,\{4\})\}, C_{34} + g(4,\{2\}$
$= \min\{13+18, 12+13\} = 25$
$d(3,\{2,4\}) = 4$

$g(4,\{2,3\}) = \min\{C_{42} + g(2,\{3\}), C_{43} + g(3,\{2\})\}$

$= \min\{8+15, 9+18\} = 23$
$d(4,\{2,3\}) = 2$

Daryl Fernandes 20200120004 (72)

(4) $|S| = 3$

$$g(1, \{2,3,4\}) = \min\{C_{12} + g(2,\{3,4\}), C_{13} + g(3,\{2,4\}), C_{14} + g(4,\{2,3\})\}$$

$$= \min\{10 + 25, 15 + 25, 20 + 23\} = 35$$

$$d(1, \{2,3,4\}) = 2$$

Finding graph:

$d(1, \{2,3,4\}) = 2$ path $1 \to 2 \to 4 \to 3 \to 1$
$d(2, \{3,4\}) = 4$
$d(4, \{3\}) = 3$
$d(3, \phi) = 1$

2) Knapsack Problem

- Knapsack problem algorithm is a very helpful problem in combinatorics. In the supermarket there are n packages ($n \leq 100$) the package i has weight $W[i] \leq 100$ value $V[i] \leq 100$.

- A theif cannot carry weight exceeding M ($M \leq 100$). The problem to be solved here is: which packages the thief will take away to get the highest

Daryl Fernandes 2020012004(72)

Input:

- Maximum weight M & & the number of packages n.
- Array of weight [i] & corresponding value V[i]

Output:

- Maximum value & corresponding weight in capacity
- Which pakages the theif will take away

Knapsack can be further divided:
i) 0/1 knapsack problem (DP)
ii) Fractional knapsack (Greedy Strategy)

| Property | Fractional Knapsack | 0/1 knapsack |
|---|---|---|
| 1) Division of object | Allowed | Not allowed. |
| 2) Optimal Solution | locally optimal | globally optimal |
| 3) Decision sequences | Single | Multiple |
| 4) Decision made | Stepwise | Not Stepwise |

Daryl Fernandes 2020012004(72)

| | | |
|---|---|---|
| 5) Approach | Greedy approach | Dynamic approach |
| 6) Time complexity | O(n log n) | O(nW)<br>n → items<br>W → weight |

Daryl Fernandes
20209120044 (72)

3)

```
M_colouring(k) {
    repeat {
        NextColor(k);
        if (x[k] = 0)
            return;
        if (k = n)
            write [x[1:n]);
        else MColouring(k+1)
    }
    until (false);
}

NextValue(k) {
    repeat {
    x[k] := (x[k] + 1) mod (m+1);
    if (x[k] = 0) then return;
    for (j := 1 to n do {
            if ((a[k,j] ≠ 0 & x[k] = x[j]))
                then break;
    }
    if (j = n+1) then return
    } until (false)
}
```

Analysis:

- Consider a given graph G has n vertices & at most m colours are available to colour it.

- In the state space tree of the decision problem, at level i, the tree has m nodes representing problem states.

Danyl Fernandes
2020012004(72)

- So, the total number of internal nodes in a state of space tree is $\Sigma_0 \; 0 \leq k \leq n-1$ $m^k$

- Therefore total no of nodes in space tree would be

$$1 + M + M^2 + M^3 \cdots + M^n$$

$$T(n) = 1 + M + M^2 + M^3 \cdots + M^n$$

$$= \frac{M^N + 1 - 1}{M - 1}$$

$$\therefore T(n) = O(M^n)$$

Daryl Fernandes
20200120004(72)

4) Algorithm :

```
Rabin_Karp_Matcher (T, P, d, q)
    n = T.length
    m = P.length
    h = d^(m-1) mod q
    p = 0
    t0 = 0
    for i = 1 to m
        p = (dp + P[i]) mod q
        t0 = (dt0 + T[i]) mod q
    for s = 0 to n-m
        if p == ts
            if P[1...m] == T[s+1...s+m]
                print "Pattern occurs with shift" s
        if s < n-m
            ts+1 = (d(ts - T[s+1]h) + T[s+m+1]) mod q
```

Analysis :

- The average & best case running time of the Rabin-Karp algorithm is $O(n+m)$ but its worst-time is $O(nm)$. Worst case of Rabin-Karp algorithm occurs when all characters of pattern & text are same as the hash values of all the substrings of txt[] match with hash value of patt[]. For example patt[] = "AAA" & txt[] = "AAAAAAA".