(1) Formulate and solve with Lindo the critical path method handout attached

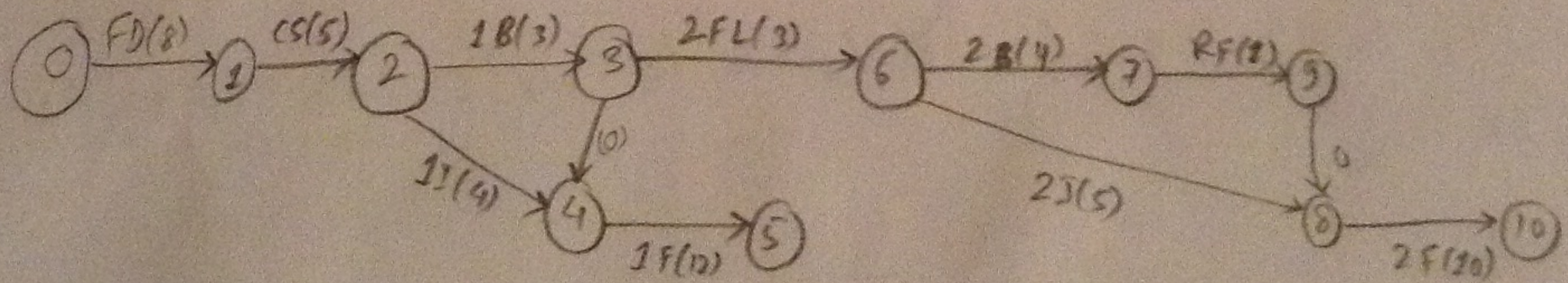
→ Construction of a small two-storey house involves the tasks listed in the following table. The table also shows the estimated duration of each task in days and the tasks that must complete before it can begin

Activity	Duration	Immediate Predecessors
FD. foundation	8	–
CS. Concrete Slab	5	FD
1B First Bearing Walls	3	CS
1I. First internal walls	4	CS
1F. First finishing	12	1B, 1I
2FL. Second floor	3	1B
2B. Second Bearing Walls	4	2FL
2I. Second internal walls	5	2FL
2F. Second finishing	10	2B, 2I, RF
RF. Roof	1	2B

To effectively administer the product development, the manager of the project would like to know the following

1) Construct a CPM project network
2) Formulate the LP to solve the CPM

1)



3)

$\min\ t_{10} - t_0$

S.t.

$t_1 - t_0 >= 8$

$t_2 - t_1 >= 5$

$t_3 - t_2 >= 3$

$t_4 - t_2 >= 4$

$t_4 - t_3 >= 0$

$t_5 - t_4 >= 12$

$t_6 - t_3 >= 3$

$t_7 - t_6 >= 4$

$t_9 - t_7 >= 1$

$t_8 - t_6 >= 5$

$t_8 - t_9 >= 0$

$t_{10} - t_8 >= 10$

$t_0 >= 0$

$t_1 >= 0$

$t_2 >= 0$

$t_3 >= 0$

$t_4 >= 0$

$t_5 >= 0$

$t_6 >= 0$

$t_7 >= 0$

$t_8 >= 0$

$t_9 >= 0$

$t_{10} >= 0$

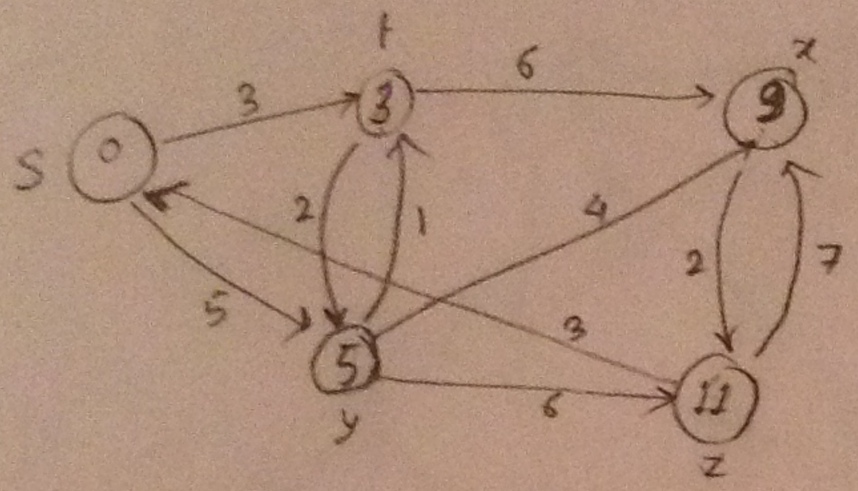
end

LP OPTIMUM FOUND AT STEP 10

OBJECTIVE FUNCTION VALUE = 34

$t_{10} = 34$, $t_0 = 0$, $t_1 = 8$, $t_2 = 13$, $t_3 = 16$, $t_4 = 17$, $t_5 = 29$, $t_6 = 19$, $t_7 = 23$, $t_9 = 23$, $t_8 = 24$

(2) Formulak and solve with Lindo the shortest path for the on pg 648 using 1 as the start node, x as end node.



min $3st + 5sy + yt + 2ty + 6tx + 6yz + 7zx + 4yx$

s.t.

$st + sy = 1$

$tx + yx + zx = 1$

$st + yt - ty - tx = 0$

$sy + ty - yt - yx - yz = 0$

$yz - zs + xz - zx = 0$

end

int 10

LP OPTIMUM FOUND AT STEP 3

Objective value = 9

$st = 1$

$sy = 0$

$yt = 0$

$ty = 1$

$tx = 0$

$yz = 0$

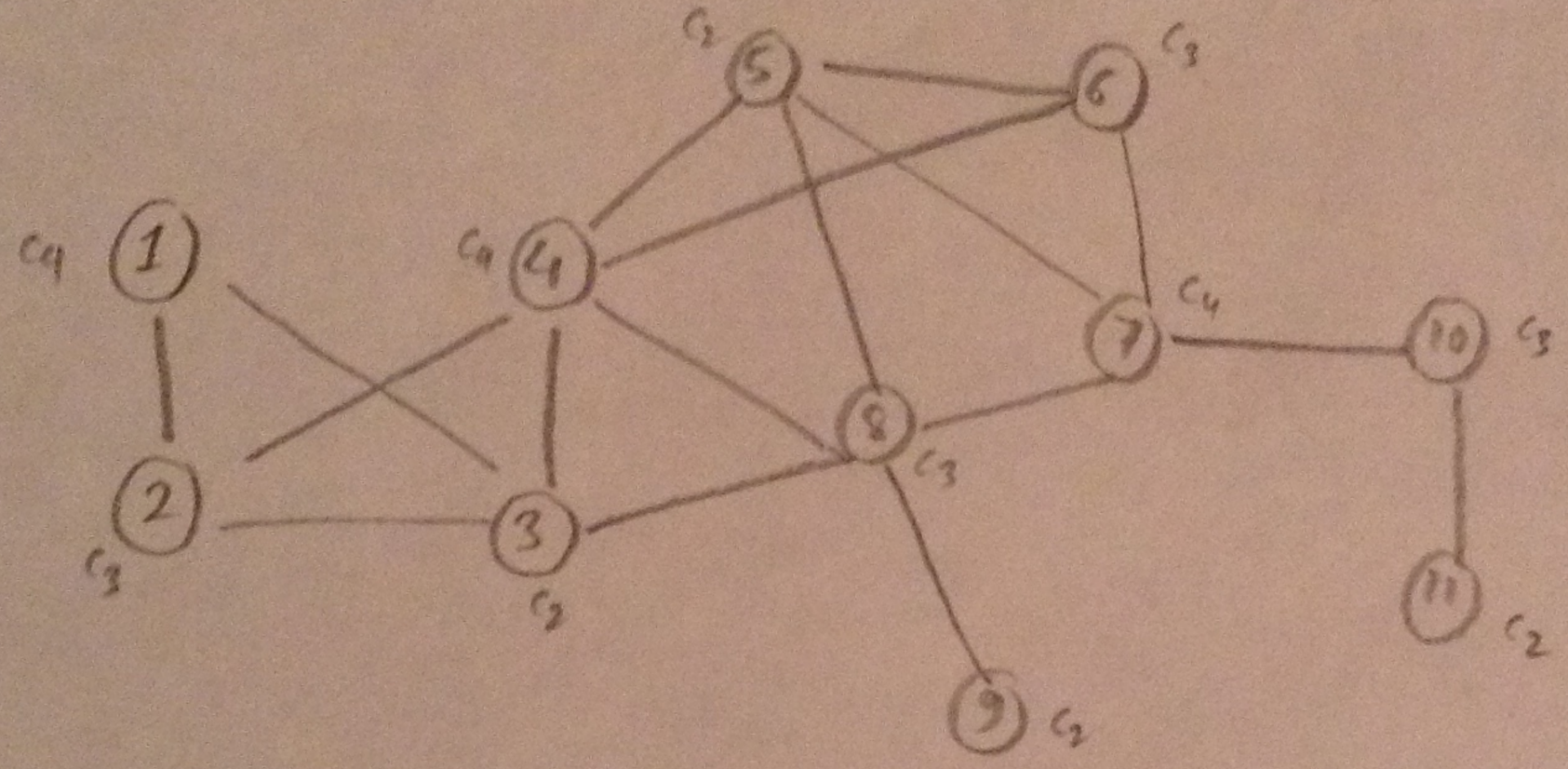
$zx = 0$

$yx = 1$

$zs = 0$

$xz = 0$

(3) Formulate to solve the coloring problem on the graph given (assume at most 6 colors), solve with Lindo

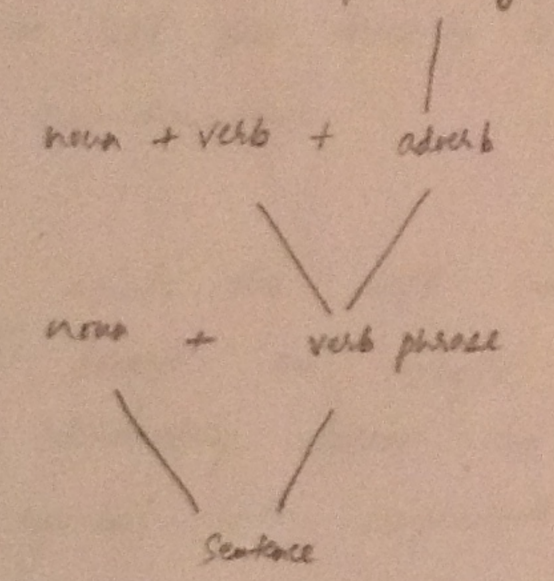


4 we defined two grammars on page 3 and page 6. Determine the type of classification for each grammar.

Ans ①

On page 3 the grammar consists of set of terminals and non-terminals
Eg sentence → noun verphrase is a definition of non-terminals

To parse Jills desires frequently the defined grammer supports LR parsing



Jills desires frequently is a valid sentence that can be parsed using LR parser

In LR parser, we reduce the collection of terminals and non terminals to non-terminal by reversing the output.

② on page 6 a LL parser is defined as

a) S → F
b) S → (S+F)
c) F → 1

To parse 1+1, we start from S and derive the sentence

S → (S+F)
S → (F+F)
S → (1+F)
S → (1+1)

In LL parser we look at leftmost symbol. If the symbol is a terminal, we match it with the sentence, if not, we use the sentence and determine next non-terminal or terminal symbol to replace it.

(5)

(a) LL parsers begin parsing at the start symbol and applies the defined rules in an approach to derive the target string, whereas LR parsers start at the target string and attempt to arrive back at the start symbol.

(b) LL parser does left-to-right leftmost derivation. It starts at the start symbol and repeatedly expands the leftmost non-terminal until the target string is derived.

LR parser does left-to-right, right most derivation. It scans the sentence from left and tries to construct the non-terminals and reach the starting non-terminal

(c) LL parser selects the left non-terminal, looks at the target string (look ahead) and based on target string's composition chose the symbols that will ultimately derive the string. It matches the leftmost terminal symbol with unparsed input symbol

Eg.

$S \rightarrow F$
$S \rightarrow (S+F)$
$F \rightarrow 1$

For string (1+1) it can parse using

$S \rightarrow [S+F) \rightarrow (F+F) \rightarrow (1+F) \rightarrow (1+1)$

LR parser (reverse) in each step adds the next input token for consideration & reduces the terminals and nonterminals with some non-terminals

Eg using above rule a LR parser would parse (1+1) as

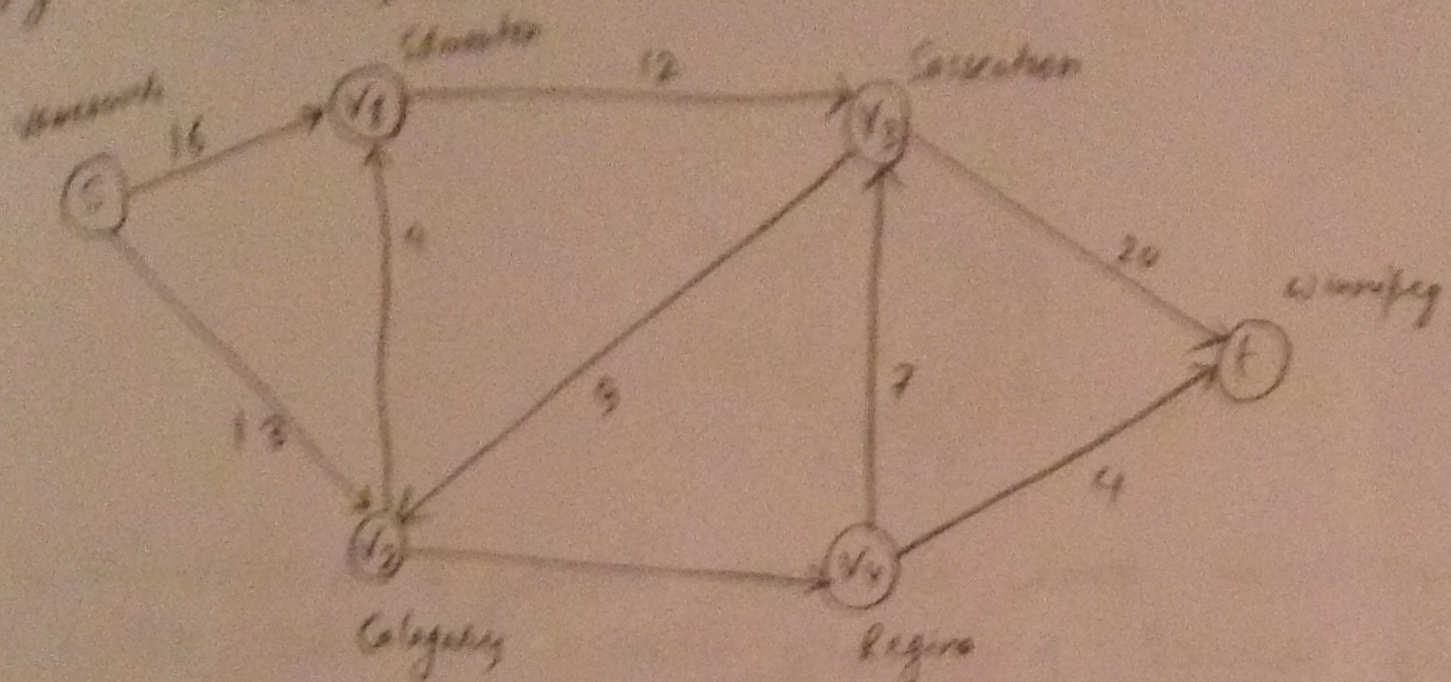
$(1+1) \rightarrow (1+F) \rightarrow (F+F) \rightarrow (S+F) \rightarrow S$

(d) LL parsers are are easier to derive by hand, but less powerful than LR parsers and only accepts small set of grammars. There are several variations to LR parsers eg SLR, LALR

(c) LL based parser called ANTLR is used in languages like Java, C# to specify lexers and parsers for various libraries and utilities

GCC uses LR based grammars which is a simplified version of LR parser called Look-Ahead LR parser (LALR parser). Tools like Yacc and bison can generate the LALR parsers.

(3) Write out explicitly the linear program corresponding to finding the maximum flow in figure 26.1(a)



A flow network $G = (V, E)$ for Lucky Puck Company's trucking problem. The vertices V is the source s, and the winnipeg warehouse is the sink t. The non-negative quantity $f(u,v)$ is the flow from vertex u to vertex v. The value $|f|$ of a flow f is defined as

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$$

i.e. The total flow out of the source minus the flow into the source

Now To maximize this

Maximize

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$$

Subject to

$$\sum_{u \in V} f(u,v) = \sum_{v \in V} f(v,u) \quad \forall u \in V - \{s,t\} \quad \text{— Flow constraints}$$

$$f(u,v) = 0 \quad \forall (u,v) \notin E \quad \text{— No flow}$$

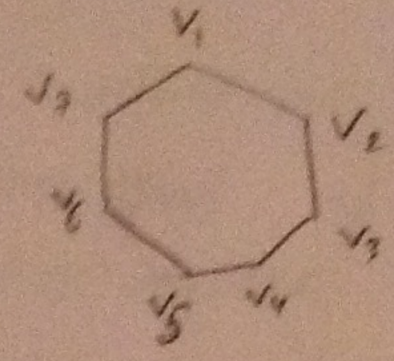
$$0 \le f(u,v) \le c(u,v) \quad \text{— Capacity constraints}$$

Using above properties in above graph, we can write following linear program as

Note: Can maximize both input ($pCV_3 + pCV_2$) or the output ($pV_3t + pV_4t$)

(1) Prove that if G is an undirected bipartite graph with an odd number of vertices, then G is non hamiltonian

Ans

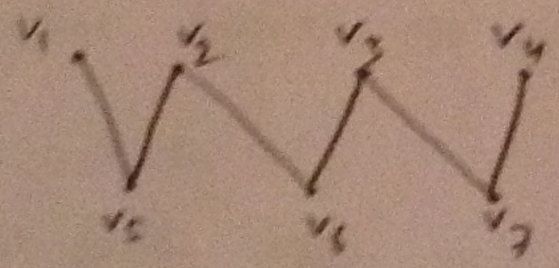


To be hamiltonian a graph needs to have hamiltonian cycle. Using above vertices, lets create a bipartite graph. A bipartite graph is a graph, whose vertices can be divided into two disjoint sets U and V, such that every edge connects a vertex in U to one in V. U and V are independent sets.

If $U = \{v_1, v_2, v_3, v_4\}$ — even nodes

$V = \{v_5, v_6, v_7\}$ — odd nodes

we can create following graph.



As you can see from above graph, there cannot be a hamiltonian graph unless v_1 connects with v_4. However in a bipartite graph v_1 and v_4 cannot connect with each other, because both of them are in U.

Hence, if G is an undirected bipartite graph with an odd number of vertices then G is non hamiltonian.

(3) Reduce 2-CNF-SAT to an efficiently solvable problem on a directed graph. We have to show that 2-CNF-SAT is solvable in linear time. Each sub expression or clause in our formula contains at most two literals.

Eg. $u \cup v$ clause is viewed as $\bar{u} \to v$ and $\bar{v} \to u$. We construct a graph such that if $a_1, a_2 \ldots$ are variables of the formula there are two vertices v_i and $\bar{v_i}$ (complement) that is in V for each a.

→ The formula is satisfiable iff no pair of complementary literals are in the same strongly connected component of G

If there is a path from a node u to v and from v to u, it implies that both the node must have same value (in truth assignment).

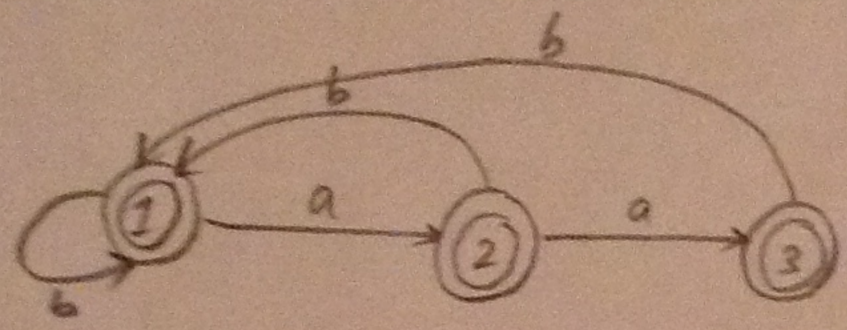
Therefore, if there is a path from u to v and from v to u following algorithm rejects the value.

```
for each a in V
    if a path from u to ū and from ū to u
            (where u v ū are possible values of a1)
        reject
    else
        accept.
```

Thus having such a formula a 2-CNF-SAT is a problem that can be solvable in polynomial time, thus it is polynomial time decidable.

10 (a) Find a regular expression for the set of strings over {a, b} that does not contain the substring aaa. Give FSM

$(b^*(ab)^* \cup (aab)^*) \cup (a \cup aa)$



$F = \{1, 2, 3\}$

(b) Every 1 followed by two 0s.

$(0^* \cup 1)(0+1)^*0^*00$

(c) Ending in 00 not containing 11

$0^*(10(0)+)^*0^*$

(d) Even no of 1s

$0^*(10^*10^*)^*$