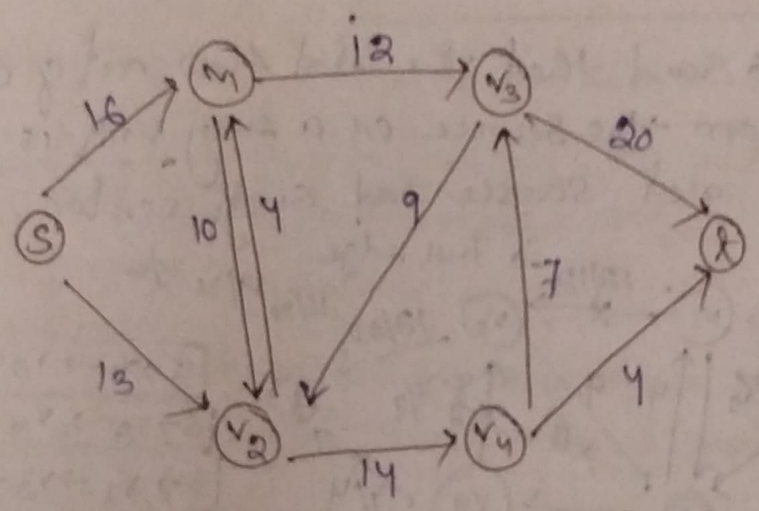
* Flow Network (G)

→ A flow network $G = (V, E)$ is a weighted directed graph in which each edge $(u, v) \in E$ has a non negative capacity $c(u,v) \geq 0$. If the edge (u,v) does not belong to E we assume that $c(u,v) = 0$.

→ A Flow network (G) has two specially designated vertices called as a source ('s') and sink ('t').

→ The objective of the flow network problem is to find a flow from the source vertex (s) to the sink vertex (t) of maximum value.

Exp



Flow ⇒ A Flow denoted as $f(u,v)$ in G is a real valued function which can be positive, zero or negative. $\boxed{f(u,v) \Rightarrow u \to v}$

→ A flow in G satisfies the following three properties

(i) capacity constraint

→ For all $u, v \in V$ we require

$$\boxed{f(u,v) \leq c(u,v)}$$

→ It is said that the flow from one vertex to another must not exceed the given capacity.

(ii) Skew symmentry

→ For all $u, v \in V$, we require

$$\boxed{f(u,v) = -f(v,u)}$$

→ It is said that the flow from a vertex u to a vertex v is the negative of the flow in the reverse direction

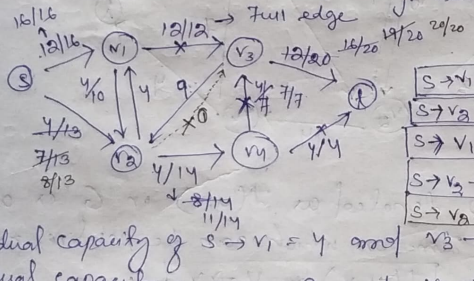
(iii) Flow conservation

→ For all $u, v \in V$, we require

$$\boxed{\begin{array}{l} u \in V - \{s, t\} \\ \text{then } \sum_{v \in V} f(u,v) = 0 \end{array}}$$

→ It ~~sais~~ said that the total flow out of a vertex other than the source or a ~~sing~~ sink is zero.

→ u is not source and sink vertex.



$\boxed{S \to v_1 \to v_3 \to t = 12}$

$\boxed{S \to v_2 \to v_4 \to t = 4}$

$\boxed{S \to v_1 \to v_2 \to v_4 \to v_3 \to t = 7}$

$\boxed{S \to v_2 \to v_4 \to v_3 \to t = 3}$

$\boxed{S \to v_2 \to v_3 \to t = 1 \ \times}$

Residual capacity of $S \to v_1 = 4$ and $v_3 \to t = 8$

Residual capacity of $S \to v_2 = 9$ and $v_2 \to v_4 = 10$

Residual capacity of $S - v_1 = 0$; $v_1 \to v_2 = 6$, $v_2 \to v_4 = 6$,

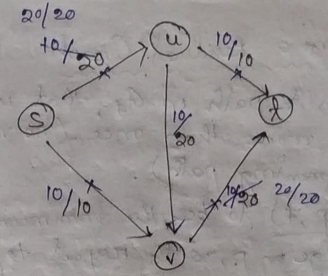
$v_4 \to v_3 = 3$ and $v_3 \to t = 4$

Then there is no path from S to t (augmented path)

so there is only 4 augmented path then

the total flow $= 23 + 1 = 24 - 1 = 23$

Exp



⇒ Residual graph (G_f)

$S \to u \to t \Rightarrow 10$

Residual capacity $S \to u = 10$

$S \to v \to t \Rightarrow 10$.

Residual capacity $S \to v = 10$ and $v \to t = 10$

$S \to u \to v \to t \Rightarrow 10$

Residual capacity $S \to u = 0$ and $u \to v = 10$ and $v \to t = 0$

There is no path from $S \to t$ so there is 3 augmented matrix and $\boxed{\text{total flow} = 30}$ and $\boxed{f(S,t) = 30}$

Properties of Flow Network

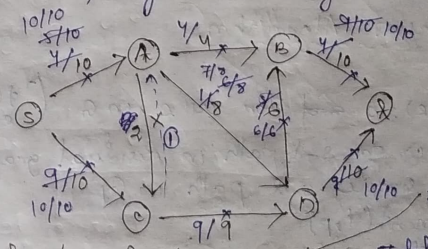
→ For a flow network there are three assumption

(i) no edge entered the source s and no edge leaves the sink t.

(ii) There is atleast one edge incident to each vertex (The degree is atleast 1)

(iii) All capacity are integers.

Exp



$S \to A \to B \to t \Rightarrow 4$

$S \to C \to D \to t \Rightarrow 9$

$S \to A \to \boxed{D} \to t \Rightarrow 1$

$S \to A \to D \to B \to t \Rightarrow 5$

$S \to C \to A \to \boxed{D} \to B \to t \Rightarrow 1$

Total flow $= 20 - 1 = 19$

we can not a backedge because there is non zero flow backward edge.

★ Augmenting path in a Residual Graph

Let p be a simple s-t path in G_f, that is p does not visit any node more than once. (no vertices are repeated in the augmenting path).

Define <u>bottleneck(p, f)</u> to be the minimum residual capacity of any edge on p, with respect to flow f.

Now, the operation augment(f, p) which yeilds a new flow f' in G is defined as follows.

```
augment(f, p)
  Let b = bottleneck(P, f)
  For each edge (u,v) ∈ P
    if e = (u,v) is a forward edge then
      increase f(e) in G by b.
    Else ((u,v) is a backward edge, and let
          e = (v,u))
      decrease f(e) in G by b.
    End if
  End for
  Return (f).
```

★ Ford-Fulkerson Algorithm

```
Max-Flow
initially f(e) = 0 and for all e in G   (f(e) = f(u,v), e = (u,v))
while there is an s-t path in residual graph G_f
  Let p be a simple s-t path in G_f
  f' = augment(f, p)
  update f to f'
  update the residual graph G_f to G_f'
End while
return (f)
```

★ Maximum flow and minimum cut (max flow and min-cut)

→ <u>cut of flow network</u>

A cut (A, B) of flow network $G = (V, E)$ is a partition of V into A and $B = V - A$ such that $s \in A$ and $t \in B$.

If f is a flow then the net flow $f(A, B)$ across the cut (A, B) is defined as

Value of the flow.

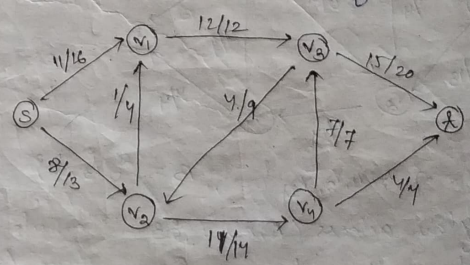
$$\boxed{\begin{aligned} f(A,B) &= \sum_{u \in A} \sum_{v \in B} f(u,v) - \sum_{u \in A} \sum_{v \in B} f(v,u) \\ &= f^{out}(A) - f^{in}(A) \end{aligned}}$$

→ The capacity of the cut(A, B) is

$$\boxed{c(A,B) = \sum_{u \in A} \sum_{v \in B} c(u,v)}$$

→ A minimum cut of the network is a cut whose capacity is minimum over all cuts of the network.

<u>Exp</u> Let us consider the instance of a residual map



SHOT ON MI A2
MI DUAL CAMERA

cut C_1. $A = \{s, v_1, v_2\}$ $B = \{v_3, v_4, t\}$

C_2 $A = \{s, v_2, v_4\}$ $B = \{v_1, v_3, t\}$

C_3 $A = \{s, v_1, v_3\}$ $B = \{v_2, v_4, t\}$

C_4 $A = \{s, v_1, v_2, v_3\}$ $B = \{v_4, t\}$

C_5 $A = \{s, v_1\}$ $B = \{v_2, v_3, v_4, t\}$

capacity of $C_1 = c(v_1, v_2) + c(v_3, v_4) = 12 + 14 = 26$

capacity of $C_2 = c(s, v_1) + c(v_4, t) + c(v_4, v_3) + c(v_2, v_1) = 31$

" of $C_3 = 13 + 9 + 20 = 42$

" of $C_4 = 14 + 20 = 34$

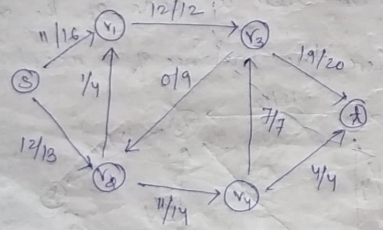
" of $C_5 = 13 + 12 = 25$ (min cut)

The net flow accross the min cut (C_5) that is

$$f(A, B) = f^{out}(A) - f^{in}(A)$$

$$= 8 + 12 - 1 = \underline{19}$$

Exp



$A = \{s, v_1, v_2, v_3\}$, $B = \{v_3, t\}$

Net flow accross $(A, B) = 23$

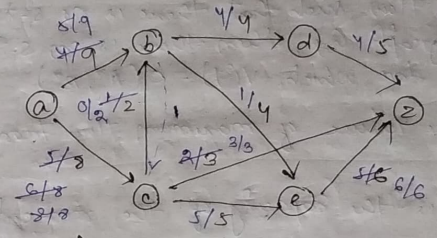
Capacity of the cut $(A, B) = 23$

Imp Note

⇒ The max-flow is always equal to min-cut.

⇒ The max-flow is equal to the sum of the flow which is towards to the sink (or) from the source vertex.

Exp



Q (i) What is the value of maximum flow from source to sink.

(ii) Show the residual graph / network G_f.

(iii) Find the mincut of the flow network and say what its capacity is?

(i) $a \to b \to d \to z \Rightarrow 4$

$a \to c \to e \to z \Rightarrow 5$

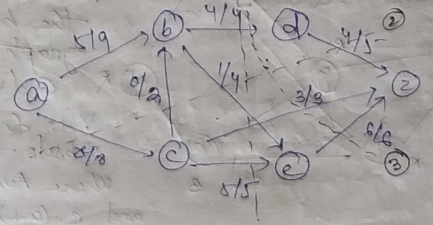
$a \to c \to b \to e \to z \Rightarrow 1$

$a \to c \to z \Rightarrow 3$

$a \to b \to c \to z \Rightarrow 1$

• max flow ⇒ 13

(ii)



(iii) ① ⇒ 15

② ⇒ 17

③ ⇒ 4 + 3 + 6 = 13

⇓ min cut

$A = \{a, b, c, e\}$

$B = \{d, z\}$

Augmenting path

$s \to x_1 \to y_1 \to t \Rightarrow 1$
$s \to x_2 \to y_2 \to t \Rightarrow 1$
$s \to x_3 \to y_3 \to t \Rightarrow 1$
$s \to x_4 \to y_5 \to t \Rightarrow 1$

max-flow = 4

The matching of the graph G is also 4 that is $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_5)$.

Hence the max-flow is equal to the max number of matching in the biparatite graph.

Co-2 NP and Computational Intractability

Tractable problem:

→ problems that can be solved in polynomial time i.e $O(n^k)$ are called Tractable problems. This is the P type problems.

Intractable problem:

→ problems that required super polynomial time are called Intractable problems.

→ Intractable problems are NP-complete type problems whose status is unknown.

Complexity class

P (polynomial)
NP (Non-deterministic polynomial)
Co-NP (Complement of NP)
NPC (NP-Complete)
NPH (NP-Hard)

→ That means NP-complete problems have computationally hard for all practical purposes. Intractable problems are computationally hard as no polynomial algorithm has yet with developed for NPC program, nor have any one yet been able to prove that no polynomial time algorithm can exist for any of them.

problem Reducibility

→ A problem y can be reduced in any problem x if any instance of y can be easily cased as a instance of x. That is if we have an algorithm capable of x then we can also solve y. (use a solution of problem x to get the sol^n of the problem y).

polynomial time reducibility.

→ If a problem y is polynomial time reducable to some problem x which can be solved in polynomial time then the problem y can also be solved in polynomial time. polynomial time reducibility of a problem y to x can be written as $\boxed{y \leq_p x}$ ("y is polynomial time reducibility to x")

Note

suppose $y \leq_p x$ then the following two statements are equivalent.

(i) If x is a solved in polynomial time then y can also be solved in polynomial time.

(ii) If y is not be solved in polynomial time then x can not be solved in polynomial time. (Contrapositive statement)

problem reduction

(i) simple problem reduction.
(ii) Hard problem reduction.

Q Closest pair to Sorting problem.

input: set of 'n' numbers and a target/threshold 'd'.

Exp $S = \{8, 4, 10, 3, 9, 15, 18, 6\} \xrightarrow{sort} \{3, 4, 6, 8, 9, 10, 15, 18\}$

$d = 2$

(3,4)
(8,9) } 1
(9,10)

(4,6)
(6,8) } 2

→ 5 sets are the closest pair.

Algorithm

```
closest-pair(S)
{
  sort(S)
  Is min_{1≤i≤n} |S_i - S_{i+1}| ≤ d
}
```

Exp $S = \boxed{8 \mid 4 \mid 12 \mid 9 \mid 3 \mid 10}$ and let $d = 1$

after sorting

1	2	3	4	5	6
3	4	8	9	10	12

so the closest pair is

(3,4), (8,9), (9,10)

Q LCM ≤p HCF

input: Two integers x, y.

output: Return the smallest integer m such that m is a multiple of x and y.

```
LCM
{
  return (xy / HCF(x,y));
}
```

Exp $LCM(24, 36) = \frac{24 \times 36}{HCF(24, 36)} = \frac{\overset{2}{\cancel{24}} \times 36}{\cancel{12}} = 72$

Q HCF ≤p LCM

input: Two integer x and y

output: Return the largest integer d such that d divides both x and y.

```
HCF(x,y)
{
  return (xy / LCM(x,y))
}
```

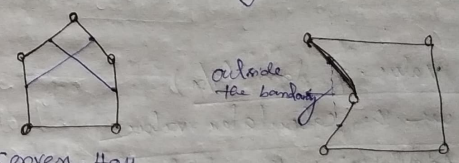
Exp $HCF(24, 36) = \frac{24 \times 36}{LCM(24, 36)} = \frac{24 \times 36}{72} = 12$

(highest common factor) (least common multiplier)

Q Sorting ≤p Convexhull

→ The convexhull of a set 'Q' of points is the smallest convex polygon 'P' for which each point in Q is either on the boundary of P or in its interior.

Exp



convex Hull — not convex Hull

→ the line is drawn from one line to another line if the line is in the polygon/boundary then it is a convexhull but otherwise they are not convexhull.

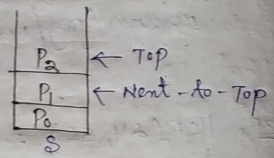
- (x,y) → rectangular coordinates
- (r, θ) → polar coordinates $\left[r = \sqrt{x^2+y^2} \quad \theta = \tan^{-1}(y/x) \right]$

Algorithm

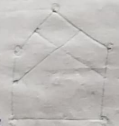
Convexhull (Q)

{

1. chose P_0 be the point in Q with the minimum y-coordinates or the leftmost such point in case of a tie.
2. Let $\langle P_1, P_2, P_3, \ldots, P_n \rangle$ be the remaining points in Q, sorted by polar angle in counter clockwise order around P_0.
3. PUSH (P_0, S)
4. PUSH(P_1, S)
5. PUSH(P_2, S)
6. for $i = 3$ to n {
7. while the angle formed by points Next-to-Top(S), Top(S) and P_i makes a nonleft turn.
   ↓ Right $\langle P_1 P_2 P_3$
8. pop(S)
9. PUSH(P_i)
   }
10. return(S)
    }



Quadrant	value of $\tan^{-1}(y/x)$
1	use the calculator value
2	Add 180° to the calculator value
3	Add 180° to the calculator value
4	Add 360° to the calculator value



→ How to test whether an angle makes left turn or nonleft turn?

Let us consider three points P, Q, and R.

For checking the angle $\angle PQR$, find the slope of this line joining the points P to Q and Q to R.

$$\boxed{\text{slope of } P \text{ to } Q \Rightarrow \frac{y_Q - y_P}{x_Q - x_P} \text{ and slope of } Q \text{ to } R = \frac{y_R - y_Q}{x_R - x_Q}}$$

$$\boxed{m = (y_Q - y_P)(x_R - x_Q) - (x_Q - x_P)(y_R - y_Q)}$$

→ if $(m > 0)$, then the angle $\angle PQR$ makes a clockwise angle i.e nonleft turn. (right turn) (↷)

→ if $m < 0$, then the angle $\angle PQR$ makes an anticlockwise angle i.e left turn. (↶)

Examples

Let the points be { (0,3), (1,1), (2,2), (4,4), (0,0), (1,2), (3,1), (3,3) }

Find the points to be on the boundary of the convexhull.

Sol

$P_0 = (0,0)$

$P_1 = (3,1)$

$P_2 = (1,1)$

$P_3 = (2,2)$

$P_4 = (3,3)$

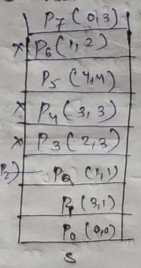
$P_5 = (4,4)$

$P_6 = (1,2)$

$P_7 = (0,3)$

→ when the angle is same then the y-coordinates are left most value.

45, 45, 45, 43.43, 18, 45



check the angle $\angle P_1 P_2 P_3$

where $P_1 = (3,1)$

$P_2 = (1,1)$

$P_3 = (2,2)$

$\frac{1-1}{1-3} = \frac{2-1}{2-1}$

$\Rightarrow m = 0(+1) - (-2)(1)$

$= +2 > 0$

(ii) $\langle P_1 P_3 P_4$ $\quad P_1 = (3,1)$

$P_3 = (2,2)$

$P_4 = (3,3)$

$$\frac{2-1}{2-3} = \frac{3-2}{3-2}$$

$m = (+1)(1) - (-1)(1) = 1+1 = 2 > 0$

Hence it is a right turn so pop the (P_3) and push P_4.

(iii) $\langle P_1 P_4 P_5$ $\quad P_1 = (3,1)$ $\quad \frac{3-1}{3-3} = \frac{4-3}{4-3}$

$P_4 = (3,3)$

$P_5 = (4,4)$ $\quad m = 2(1) - 0(1)$

$= 2 > 0$

pop(P_4) and push(P_5)

(iv) $\langle P_1 P_5 P_6$ $\quad P_1 = (3,1)$ $\quad \frac{4-1}{4-3} = \frac{2-4}{1-4}$

$P_5 = (4,4)$

$P_6 = (1,2)$ $\quad m = 3(-3) - (1)(-2)$

$= -9 + 2 = -7 < 0$

Hence not pop(P_5) and only push(P_6)

(v) $\langle P_5 P_6 P_7$ $\quad P_5 = (4,4)$ $\quad \frac{2-4}{1-4} = \frac{3-2}{0-1}$

$P_6 = (1,2)$

$P_7 = (0,3)$ $\quad m = (-2)(-1) - (-3)(1)$

$= +2+3 = 5 > 0$

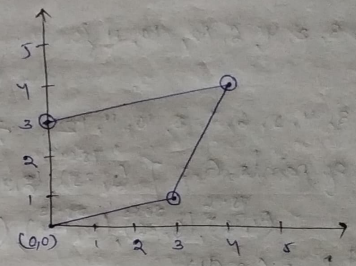
pop(P_5) and push(P_7)

(vi) then terminate the for loop

so the stack is P_7, P_5, P_1, P_0 (LIFO)

points of the stacks are $P_0(0,0)$, $P_1(3,1)$, $P_5(4,4)$ and $P_7(0,3)$. Ans.

All these points will lie on the boundary of the convex hull



Sorting $\Leftrightarrow$ convex hull ($O(n \log n)$)

input: A set S of n integers.

output: Find the smallest convex polygon containing all points created from S.

→ here we shall transfer from sorting to convex hull.

→ this means we must translate each number to a point. we do so by mapping the number 'x' to the point (x, x^2).

→ that means each integer is mapped to a point on the parabola $y = x^2$.

→ since this parabola is the convex, every point must be on the convex hull.

→ since neighbouring points on the convex hull have neighboring 'x' values, the convex hull returns the points sorted by the 'x'-coordinate, that is the original numbers.

Algo

Sort(S)

for each $i \in S$, create point (i, i^2) ⟶ $O(n)$

call subroutine convex-hull on this point set. ⟶ $(O(\log n))$

from the left most point in the hull read off the points from left to right. ⟶ $O(n)$

Exp

$S = \{5, 2, 7, 3, 8, 4, 6, 12, 10, 11\}$

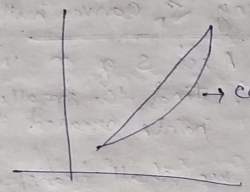
SORT(S)

so $S = \{2, 3, 4, 5, 6, 7, 8, 10, 11, 12\}$ → (objective)

create a set of points $G = \{(5,25), (2,4), (7,49), (3,9), (8,64), (4,16), (6,36), (12,144), (10,100), (11,121)\}$

78.69, 81.86, 71.36, 82.37, 75.96, 86.53, 85.23, 84.28, 84.80

$P_0 = (2,4)$	$P_6 = (8,64)$
$P_1 = (3,9)$	$P_7 = (10,100)$
$P_2 = (4,16)$	$P_8 = (11,121)$
$P_3 = (5,25)$	$P_9 = (12,144)$
$P_4 = (6,36)$	
$P_5 = (7,49)$	



The 'x' value are sorted.

Vertex cover $\leq_p$ Independent set

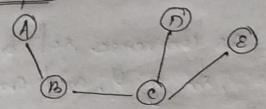
→ A vertex cover of an undirected graph $G = (V, E)$ is a subset of Vertices $V' \subseteq V$ such that if an edge $(u,v) \in E$ then either $u \in V'$ or $v \in V'$ or both u and $v \in V'$.

→ That is each vertex covers its incident edges, and a vertex cover for G is a set of vertices that covers all edges in E.

→ In vertex cover problem if a Graph G is given, we have to find minimum number of vertices that cover all edges in G.

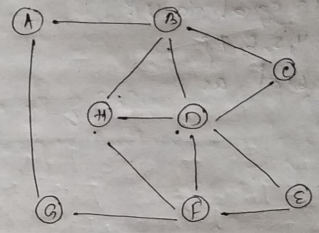
Example :-

(1)



$V_c = V' = \{B, C\}$ or $\{A, C\}$ (minimum size)

(2)



$V' = V_c = \{A, B, D, F\}$,

$\{B, D, G, F\} = \{D, B, G, F\}$

$S = \{A, C, H, E\}$

$V' = V - S = \{B, D, G, F\}$

input: A graph $G = (V, E)$ and integer $K \leq |V|$

output: A subset V' of atmost K vertices such that every $e \in E$ contains atleast one vertex in V'.

Independent set of vertices (maximum Independent set)

→ A set of vertices 'S' of a graph 'G' is independent if there are no edges (x, y) where both $x \in S$ and $y \in S$. this means there are no edges between any two vertices in independent set

$$\boxed{V = V' + S}$$

where

V = set of the vertices.

V' = vertex cover

S = Independent set.

algo

```
Vertex Cover (G, K)
{
  G' = G
  K' = |V| - K
  return Independent set (G', K')
}
```

(where K' = independent vertices, K = vertex cover)

Venten-Coven Vp Set-coven

Setcoven → Given a set U called as universe set, and a collection of subset $s_1, s_2, ---, s_m$ of U, and an integer K.

→ Determine whether there exists K or fewer ~~sbu~~ subsets such that their union is equal to U.

Example let $U = \{1, 2, 3, 4, 5, 6, 7\}$

$K = 2$ and 6 subset as

$s_1 = \{1, 2\}$

$s_2 = \{3, 4, 5, 6\}$

$s_3 = \{3, 7\}$

$s_4 = \{2, 4\}$

$s_5 = \{5\}$

$s_6 = \{1, 2, 6, 7\}$

Find a setCoven for U.

$\boxed{\text{set cover} = s_2 \cup s_6}$

Construction of reduction

Given $G = (V, E)$ and K as input to venten-coven.

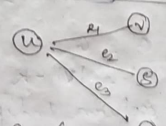
For setcoven let us use input as:

(1) $U = E$, The universe set.

(2) For each venten $u \in V$ create a subset

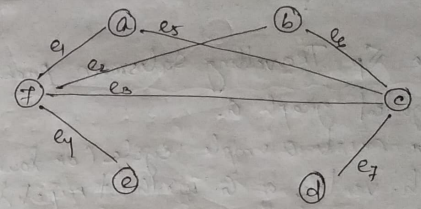
$s_u = \{ e \in E \mid e \text{ incident to } u \}$

(3) $K' = K$.



$s_u = \{e_1, e_2, e_3\}$

Example



→ graph G.

and $G = (V, E)$

$K = K' = 2$

i.e almost 2 setcoven that is equal to U.

(i) $U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ (create a universal set (U))

(ii) create a subset for each venten

$s_a = \{e_1, e_5\}$

$s_b = \{e_2, e_6\}$

$s_c = \{e_3, e_5, e_6, e_7\}$

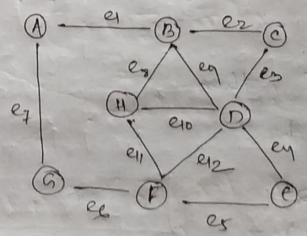
$s_d = \{e_7\}$

$s_e = \{e_4\}$

$s_f = \{e_1, e_2, e_3, e_4\}$

$s_c \cup s_f = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} = U$

Example



venten Coven (G, K) and $K = 4$

$U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$

Setcoven

$s_A = \{e_1, e_7\}$

$s_B = \{e_1, e_2, e_8, e_9\}$

$s_C = \{e_2, e_3\}$

$s_D = \{e_3, e_4, e_9, e_{12}\}$

$s_E = \{e_4, e_5\}$

$s_F = \{e_8, e_5, e_{11}, e_{12}\}$

$s_H = \{e_{10}, e_{11}, e_8\}$

$s_G = \{e_6, e_7\}$

$\boxed{s_B \cup s_D \cup s_G \cup s_F = U}$

Vertex cover = { B, D, G, F }

Hamiltonian Cycle & Travelling salesman problem

Input: An un-weighted graph G.

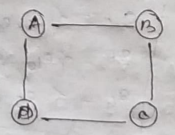
output: Does there exists a simple cycle (or tour) that visits each vertex of G without repetition?

input: An weighted complete graph G.

output: Does there exists a tour (cycle) of the salesman with minimum cost/distance without repeating the vertex.

```
Ham-cycle(G)
{
  Construct a complete weighted graph G* = (V*, E*),
    n = |V|                          where V* = V.
  for i = 1 to n
      for j = 1 to n
          if (i,j) ∈ E, then
              w(i,j) = 1
          else
              w(i,j) = 2
  return TSP(G*, n)
}
```

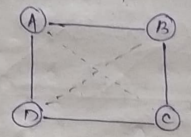
Example



$V = \{A, B, C, D\}$

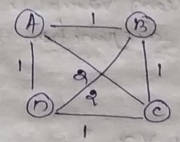
$E = \{(A,B), (A,D), (B,C), (C,D)\}$

(i) create a complete graph $G^* = (V^*, E^*)$



$(V^* = V)$

(ii) Assigned the value 1 to the edges which are present in the previous graph. else assign the value 2 to the remaining edges.



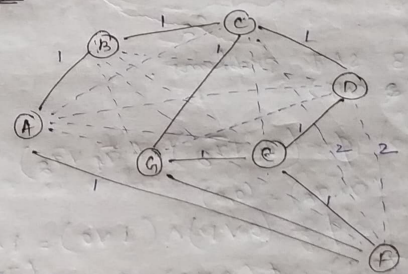
$A \to B \to C \to D \Rightarrow$ cost is 4

Note

→ The graph (G) is always connected graph.

→ The cost is get from the G^* (complete graph) the no. of vertices.

Example



cost of the edge (dotted edge) = 2

$A \to B \to C \to D \to E \to G \to F \to A \Rightarrow 1+1+1+1+1+1+1 = 7$

Satisfiability problem (SAT problems)

- Formula-SAT (ϕ)
- Circuit-SAT (ckt-SAT)

→ SAT is a decision based problem.

→ There is two value that satisfies the formula. i.e literal (0, 1) or boolean variable (true, false)

→ clauses are defined as the disjunction to the boolean variable.

$$\phi = \underbrace{(x_1 \vee \bar{x}_2)}_{c_1} \wedge \underbrace{(\bar{x}_1 \vee x_2)}_{c_2}$$

so clauses (c_1 and c_2) are consists of 2 variables

Hence there is 2 SAT problem. (2 CNF)

$$\phi = \underbrace{(x_1 \vee x_2 \vee \bar{x}_3)}_{c_1} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}_{c_2}$$

so there is 3 SAT problems. (2 CNF)

Exp Let

x_1	x_2
0	0
0	1
1	0
1	1

$\phi = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2)$

$(x_1, x_2) = (0,0)$

$\phi = (0 \vee 1) \wedge (1 \vee 0) = 1 \wedge 1 = 1$ (True) (satisfy)

$(0,1) \Rightarrow \phi = (0 \vee 0) \wedge (1 \vee 1) = 0 \wedge 1 = 0$ (false) (not satisfy)

$(1,0) \Rightarrow \phi = (1 \vee 1) \wedge (0 \vee 0) = 1 \wedge 0 = 0$ (not satisfy)

$(1,1) \Rightarrow \phi = (1 \vee 0) \wedge (0 \vee 1) = 1 \wedge 1 = 1$ (satisfy)

$\phi = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$

Let

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

~~(0,0,0) ⇒ ϕ =~~

Ans

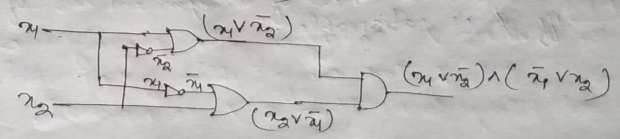
x_1	x_2	x_3	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	a: $x_1 \vee x_2 \vee \bar{x}_3$	b: $\bar{x}_1 \vee \bar{x}_2 \vee x_3$	$a \wedge b$	Result
0	0	0	1	1	1	1	1	1	satisfy
0	0	1	1	1	0	0	1	0	not satisfy
0	1	0	1	0	1	1	1	1	satisfy
0	1	1	1	0	0	1	1	1	satisfy
1	0	0	0	1	1	1	1	1	satisfy
1	0	1	0	1	0	1	1	1	satisfy
1	1	0	0	0	1	1	0	0	not satisfy
1	1	1	0	0	0	1	1	1	satisfy

→ SAT is a boolean formula (ϕ) ~~or a circuit~~ with a boolean variables and clauses

Conjuctive normal form (CNF)

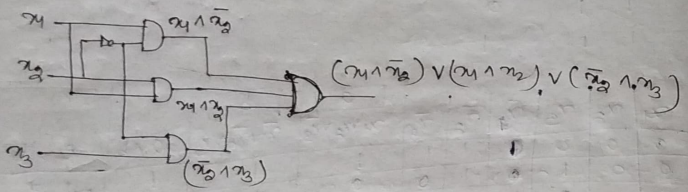
When an expression is in the form $c_1 \wedge c_2 \wedge c_3 \wedge \ldots \wedge c_n$ where c_i's are the clauses for $i = 1, 2, 3, \ldots n$. The expression is in CNF.

Circuit SAT (ckt-SAT)



x_1	x_2	$\bar{x}_1$	$\bar{x}_2$	$x_1 \vee \bar{x}_2$	$\bar{x}_1 \vee x_2$	$\phi = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2)$
0	0	1	1	1	1	1 (satisfy)
0	1	1	0	0	1	0 (×)
1	0	0	1	1	0	0 (×)
1	1	0	0	1	1	1 (satisfy)

Exp-2



x_1	x_2	x_3	$\bar{x}_2$	$x_1 \wedge \bar{x}_2$	$x_1 \wedge x_2$	$\bar{x}_2 \wedge x_3$	ϕ	satisfy
0	0	0	1	0	0	0	0	×
0	0	1	1	0	0	1	1	✓
0	1	0	0	0	0	0	0	×
0	1	1	0	0	0	0	0	×
1	0	0	1	1	0	0	1	✓
1	0	1	1	1	0	0	1	✓
1	1	0	0	0	1	1	1	✓
1	1	1	0	0	1	0	1	✓

3-SAT

→ If each clause in a given boolean formula contains exactly 3-distinct literals so it is said to be 3-SAT.

3-SAT $\leq_p$ Independent set of vertices

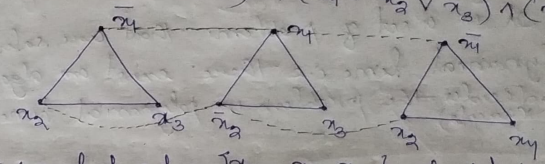
Lemma: Given an instance ϕ of 3-SAT, we construct an instance (G, K) of independent set of size K iff ϕ is satisfiable.

(i) Construction Steps:

(a) G contains 3 vertices of each clause one for each literal.
(b) Construct 3 literals in a clause in a triangle.
(c) Connect literals to each of its negations.
(d) Take K be the number of clauses.

Example

Let $\phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$



Independent set = $\{x_3, x_1, x_4\}$ should be equal to the number of clauses in the formula.

→ If the variable of Independent set is 1 then the each clause is satisfy and the overall formula (ϕ) is satisfy.

Complexity classes

(i) polynomial (P)
(ii) Non-deterministic polynomial (NP) } → if yes
(iii) complement of NP (Co-NP) } → then NO
(iv) NP-complete (NPC)
(v) NP-Hard (NPH)

Definations

Acceptability

→ A problem is accepted in polynomial time by an algorithm (A) if it is accepted by 'A' in time $O(n^k)$ for a constant k.

Decidability

→ A problem is decided in polynomial time by an algorithm 'A' if there is a constant 'k' such that for any length 'n' string in the algorithm 'A' correctly decides whether $x \in L$.

in time $O(n^k)$.

Verifiability

An algorithm with two arguments (one is the input string x and other is a certificate y) verifies an input string x if there exists a certificate such that $A(x,y) = 1$.

Class - P :- It is a set of problems that are solvable in polynomial time i.e there exist an algorithm 'A' that (decides / accepts) the problem in polynomial time.

Class - NP

→ The class of decision problems that are solvable in polynomial time on a non-deterministic algorithm.

→ That is there exist an algorithm that will verify the problem in polynomial time.

→ For example → SAT and TSP (Travelling salesman problem)

Class - NPC

→ A problem is NPC if

(i) it is in NP.

(ii) Every problem in NP reduces to it. That is a problem X is NP-complete if

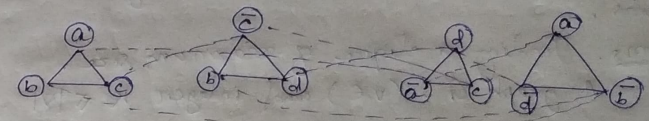
(a) $X \in NP$.

(b) $y \leq_p X$ for every problem $y \in NP$.

Class - NPH

→ If the problem X satisfy only the property i.e $y \leq_p X$ for every problem $y \in NP$ but X is not belongs to NP then X is NP-Hard.

Exp Let $\phi = (a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$



Independent set = $\{a, b, \bar{c}, \bar{d}\}$

Algo

```
3-SAT(φ)
{
  k = number of ~~clauses~~ clauses in φ.
  For each (clause (x1 ∨ x2 ∨ x3) in φ)
    create a cluster (i.e triangle) consisting of three
  vertices labeled with x1, x2, and x3.
  create edges (x1, x2), (x2, x3), (x3, x1) between all pairs
  of vertices in the triangle.
  For each (vertex xi)
    create edges between xi and all its complement
  vertices x̄i.
  return Independentset(G, k)
}
```

MID SEM

upto - chapter - 10

→ Extending the limits of intractability. (10.1)

→ Chapter 9 : pspace : A class beyond NP (9.1, 9.2, 9.3, 9.4, 9.5).

Clique problem (It is NP problem)

→ A clique is a complete subgraph where each vertex pair has an edge between them.

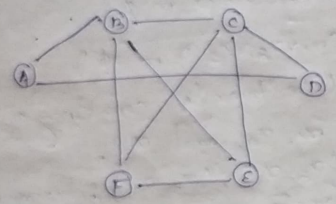
* the size of the clique is the number of vertices it contains.

Problem: Finding a clique of maximum size.

input: A graph $G = (V, E)$ and integer $K \leq |V|$.

output: A clique of size K vertices (i.e to check, is there a subset $S \subset V$ where $|S| \leq K$ such that every pair of vertices in S defines an edge in G.)

Example

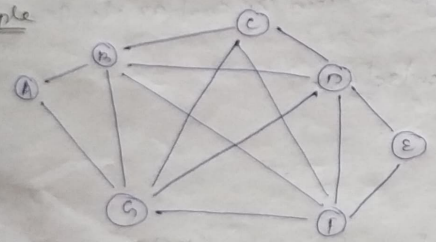


$G(V, E)$

clique : $\{B, C, E, F\}$

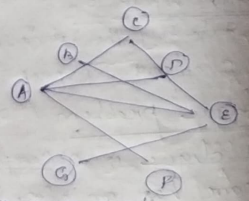
size of clique = 4.

Example



clique : $\{B, C, D, F, G\}$

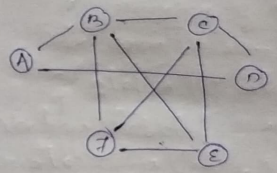
size of clique = 5



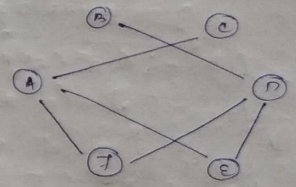
Independent set = $\{B, C, D, F, G\}$

Clique & Independent Set

```
clique (G, K)
{
construct a graph G' = (V', E')
where V' ∈ V
For all edges (i, j) ∉ E
    add an edge (i, j) to E'
return Independent_set(G', K)
}
```



G



(complement G^x)

Independent ~~size~~ set of complement Graph $G^x = \{B, C, E, F\}$

→ If there is an Independent set of size K in the complement graph (G'/G^x) it implies that no two vertices share one edge in G' which further implies all of those vertices share an edge with all others in G forming a clique. That is there exists a clique of size K in G.

Some NP-complete problem

(1) circuit-SAT
(2) 3-SAT
(3) TSP
(4) Hamiltonian cycle
(5) Vertex cover
(6) set cover
(7) Independent set.
(8) clique.
(9) Hamiltonial path.
~~(10)~~