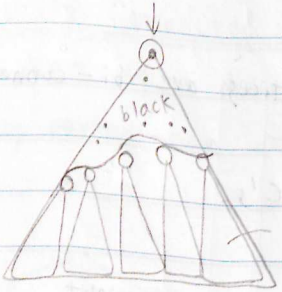


for j=1 to m do c[0,j] = j * S_1 j

4. for i=1 to n do
   for j=1 to m do

$$c[i,j] = \max \begin{cases} c[i-1,j-1] + score[a_i, \\ c[i-1,j] + S_1 \\ c[i,j-1] + S_1 \end{cases}$$

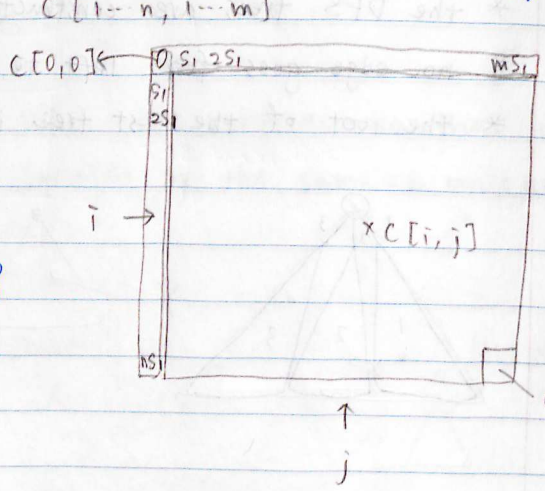
Nov-5-2015

Alignment

S_1 : ——i——

S_2 : ——j——

How similar are S_1 and S_2 ?



① decompose large instance to small one

② use small instance several times

match scores

0 < S_2 = matched

0 > S_1 = matched with "—"

0 > S_0 = symbol mismatch

Find an alignment that maximizes the total score.

c[i,j] = the better match of $a_1 \dots a_i$ and $b_1 \dots b_j$

S_1 : $[a_1\ a_2 \dots\dots a_{i-1}\ a_i] \dots\dots a_n$ (i)

S_2 : $[b_1\ b_2 \dots\dots b_{j-1}\ b_j] \dots\dots b_m$ (j)

Compute c[i,j]

$$c[i,j] = \begin{cases} \text{match } a_i \text{ \& } b_j \\ \text{match } a_i \text{ \& "—"} \\ \text{match "—" \& } b_j \end{cases}$$

$$\begin{cases} a_1 \dots a_{i-1} & a_i \\ b_1 \dots b_{j-1} & b_j \end{cases} \rightarrow = c[i-1, j-1] + score[a_i, b_j$$

$$\begin{cases} a_1 \dots a_{i-1} & a_i \\ b_1 \dots b_j & - \end{cases} \rightarrow = c[i-1, j] + score[a_i, -$$

$$\begin{cases} a_1 \dots a_i & - \\ b_1 \dots b_{j-1} & b_j \end{cases} \rightarrow = c[i, j-1] + score[-, b_j$$

Max-Matching

G: undirected graph

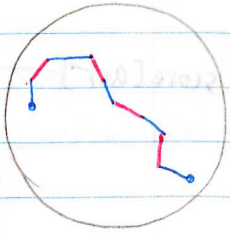
M: a matching in G (which is a collection of edges that share no common ends)

Problem: given an undirected graph G, construct a max-matching in G

Def M = a matching in G

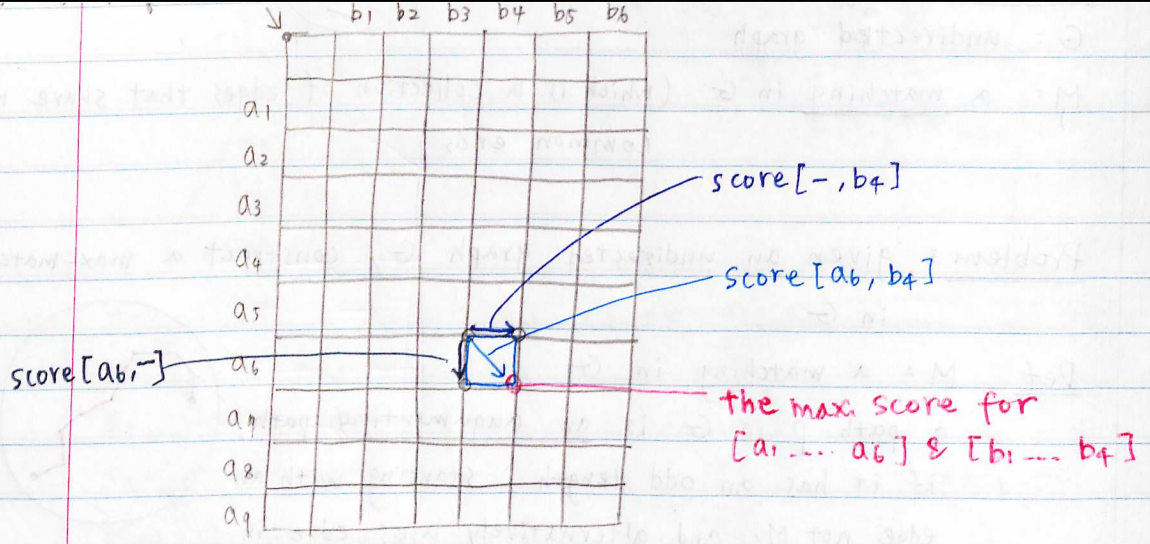
a path p in G is an augmenting path if it has an odd length, starting with an edge not M, and alternatively uses edge in M and not in M

and the two ends of p are not matched



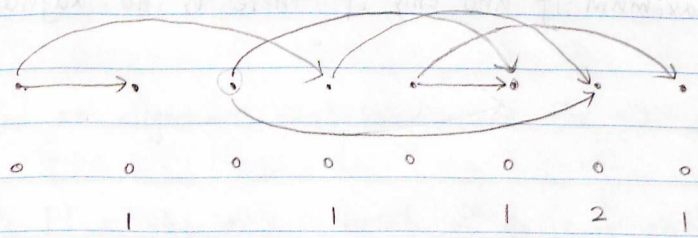
Thm M: a matching in G

M is maximum if and only if there is no augmenting path.



topological sorting → DFS

How to find the longest path for an acyclic graph ?



3. if pair = ϕ
then report(ϕ)
else
report(pair)

color[v] = black;

Nov-10-2015 HW#4, Q#2 :

* adjacency matching

DFS(v) : value

1. color[v] = gray;

1.1 pair = ϕ ;

2 for each edge [v, w] do

if color[w] == white

then DFS(w)

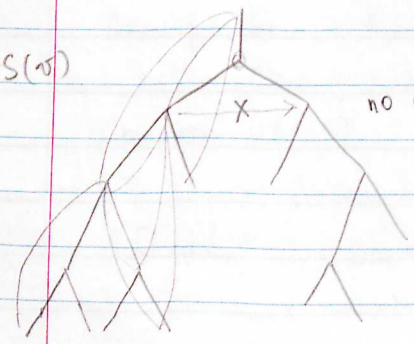
else if color[w] == gray

then if pair ≠ ϕ [v, w]

then match the edge in pair with

else

then pair ← [v, w]



then V = DFS(w)

if V = [w, w2]

then pair[w, w2]

with [v, w]

else pair ← [v, w]

M : a matching (a set of edges sharing no common ends)

Graph - Matching : given undirected graph, find a max. matching.

When G, M are given, a vertex is <u>matched</u> if it is an end of an edge in M.

Augmenting path : starting and ending at unmatched vertices, goes alternatively between unmatched and matched edges.

<u>Thm</u> G, M,
M is maximum if and only if there is no augmenting path.

pf: if there is an aug. path, then M is not max.

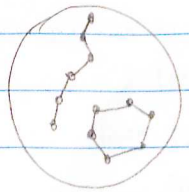
if M is not max, then there is an aug. path.

M M_{max} $|M| < |M_{max}|$

- look at the graph $M \otimes M_{max}$ (exclusive OR) = { consists of edges that belong to exactly one of M and M_{max} }

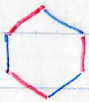
every vertex has at most degree 2

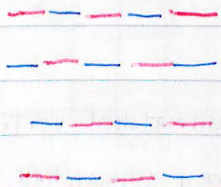
- vertex-deg in $M \otimes M_{max} \leq 2$
- every piece of $M \otimes M_{max}$ is either a cycle or a path



$M \otimes M_{max}$

every piece of M ⊕ Mmax is either a cycle or a path.

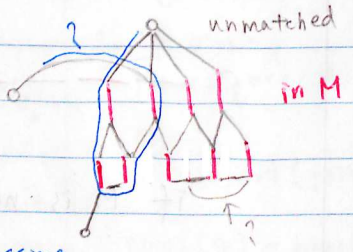




Algo:

1. $M = \{e\}$;
2. <u>while</u> there is an any path p w.r.t. M <u>do</u>
   swap edge in P and M to get a larger matching M

How to find an aug. path ?



G: bipartite
if the vertices of G can be partitioned
into two parts such that all edges are crossing

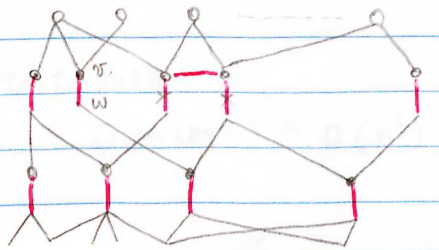
Nov-12-

Max-B Matching

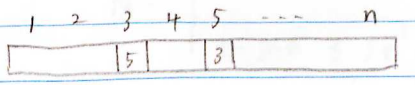
given a bipartite graph G, construct a max matching in G

Thm a matching M in G is not max if and only if there is an argumenting path.

level



matching M



other possibilities:

$$level[w] = \begin{cases} level[v]+1 & \text{can be ignored} \\ level[v] - \underbrace{a}_{2K} & \text{impossible} \\ level[v] - 2k-1 & \text{can be ignored} \end{cases}$$

Algo: time = O(m+n)

Input (G, M)

1. for each vertex v do
   level[v] = -1;
2. for each unmatched vertex v do
   level[v] = 0;
   BFS (FIFO) queue Q ← v;
3. while Q ≠ ∅ do
   v ← Q;

case 1: level[v] = odd
Let [v, w] be in M;
if level[w] = -1
then Dad[w] = v;
level[w] = level[v] + 1;
Q ← w;
else ∥ level[w] = level[v]
Stop with an aug. path

case 2: level[v] = even
for each edge [v, w] do
if level[w] = -1
then Dad[w] = v;
level[w] = level[v] + 1;
Q ← w;

else if level[w] == level[v]
stop with an aug. path

Matching (G) time = $O(nm)$

1. $M = \{e\}$ // aug. edge e
2. while there is an aug. path p do
   make a larger matching M

for bipartite graph
best algo: $O(m\sqrt{n})$ ← square root of n

for general graph: $O(n^5)$
$O(n^4)$
$O(n^3)$
$O(m\sqrt{n})$

Nov-17-2015

Shortest path :

G, s, t → a shortest path

Longest path :

G, s, t → a longest path

$\mathbb{P}$: the set of all "easy" problems

- A Problem Q is soluable in polynomial time (i.e. in $\mathbb{P}$) if it is solved by an Algorithm A that, an any instance x of length n in Q, runs in time $O(n^c)$, where c is a fixed constant.

Shortest-path

$$O(m \log n) = O(n^3) = O(N^3) \quad \begin{pmatrix} \log n \le n \\ m \le n^2 \end{pmatrix}$$

$$N = \begin{cases} n^2 \to n^3 & \text{adj. matrix} \to \begin{cases} \text{① each number has no more than } n \text{ bits} \\ \text{② there are } n \times n \text{ numbers} \\ \Rightarrow \text{① + ②} = n^3 \end{cases} \\ n + 2mn \le 3n^3 & \text{adj. list} \end{cases}$$

Independent Set (IS)

given a graph G, find a largest set of vertices in which no two are adjacent

→ no polynomial time is known

Best alg. for IS runs in time $O(1.201^n)$ — exponential time

It is believed this is difficult, i.e., it has no p-time alg.

↑ build a system to show this

IS-D Decision Problem

given a graph G and an integer K, is there an independent set of K vertices? Only need a yes/no answer

If IS-D is solvable in p-time, then so is IS.

Suppose IS-D is solved by alg. A_{ISD} of time $O(n^c)$, we want to solve IS.

Alg A1

runs in time $O(n^{c+1})$

Input G // wanted max IS

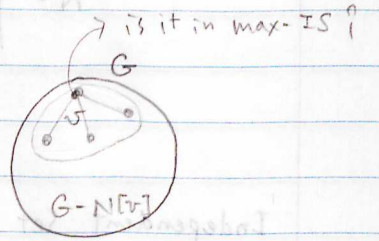
1. K = 0 $\quad k \le n$
2. loop
   call $A_{ISD}(G,K)$; for VCD
   if this is a No, then exit; YES → K = K-1] what is the largest size of the IS
   else K = K+1; → K = K+1
3. K = K-1 // K is the size of max-IS
4. return (K)

Alg A2 $\quad$ time = $O(n^{c+3})$

Input G

$O(n^{c+1})$ → 1. K = A1(G);

2. for each vertex v in G do ← $n \cdot n^{c+1} \cdot n$
   $K_1 = A1(G - N[v])$; ← self & neighbors
   if $K_1 = K-1$
   then include v in the output;
   $G = G - N[v]$; → $G = G - V$
   $K = K-1$;

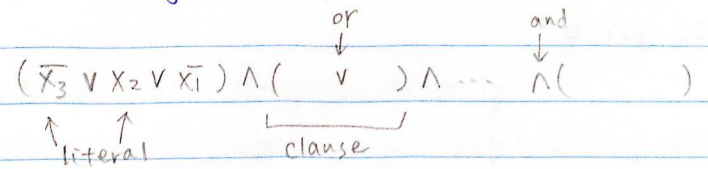


given a graph G and K, is there a set C of K vertices such that every edge has at least one end in C?

Satisfiability (SAT)

conjuncted normal form

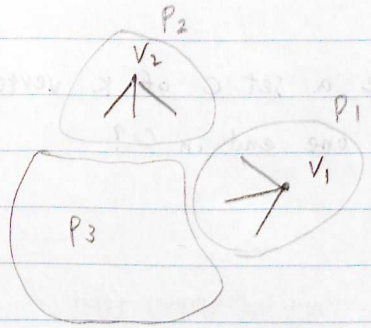
given a boolean formula F in CNF form, decide if there is an assignment that makes F true.



Partition

given n integers $a_1, \ldots, a_n$, can they be partitioned into two parts of equal sum?

Nov-19-2015



size

1. M-IS for $P_2 + P_3 \neq k-1$
2. M-IS for $P_1 + P_3 = k-1$ — contradict
3. M-IS for $P_3 = k-2$

M-IS for $P_3 \cup \{v_2\} = k-1$

$P_2 + P_3$

Step ① $v_1 \notin$ M-IS for G

② $v_2 \in$ M-IS for G

Algo A2

Max-IS(G)

1. k = maxSize(G);
2. let the vertices of G be $v_1, v_2 \ldots, v_n$;
3. for $i = 1$ to n do

   if v_i is in G & $k-1$ = maxSize$(G - N[v_i])$

   then output(v_i);

   $G = G - N[v_i]$;

   $k = k-1$

IS Does a given graph G have an independent set of K vertices?

VC Does a given graph G have a vertex cover of K vertices?

SAT Is a given CNF formula F satisfiable?

Partition Can a set S of n integer be partitioned into two parts of equal sum?

☆ Above four has common property =
Evidences to support us say yes, which is easy to check
check the evidences
if ans. is YES, some evidences make it to be yes
if ans. is NO, no way to make a assignment to make it yes

☆ These problems have the following property:
① for each yes-instance, there are evidences that are sufficient to convince me to say "yes"

② for each no-instance, there is nothing that would convince me to say "yes"

We say that a decision problem is in NP if there is an algo. A such that:
1. for each "yes"-instance X, there is a y and $A(x,y) = 1$

2. for each "no"-instance x, for all y, $A(x,y) = 0$

3. A runs in time polynomial in $|x|$

1. $S_1 \cap S_2 \neq \phi \rightarrow$ no
2. $S_1 \cup S_2 \neq S \rightarrow$ no
3. $sum(S_1) \neq sum(S_2) \rightarrow$ no
4. return yes

not n^c

IS is in NP

// an instance of IS is $X = (G, K)$

Consider the following algo

A ($X = (G, K)$, y) // $y = (S_1, S_2)$, $S_1 \& S$ or $S_2 \& S \rightarrow$ no

1. if (y is not a list of K vertices in G ; subset of S) then return ('no') // if $y \neq \phi$
2. if two vertices in y are adjacent in G then return ('no') // S_1 or $S_2 \neq \phi$
3. return ('yes')

// 2. $S_1 \cap S_2 \rightarrow$ no, $S_1 + S_2 \neq S \rightarrow$ no
// 3. $sum(S_1) \neq sum(S_2) \rightarrow$ no

Check = if (G, K) is a "yes", then let y be the list of K vertices that makes an independent set

if $X = (G, K)$ is a "no", any case will not pass step1 or step

Thm : Let Q be a decision problem, Q is in P, then Q is in NP

$P \subseteq NP$ ⇒ easy problems are also in NP

↑ subset of problems that are soluable in polynomial-time

$P \stackrel{?}{=} NP$ ⇒ still unknown

Most people believe that $P \neq NP$

⇒ some problems in NP are possibly hard

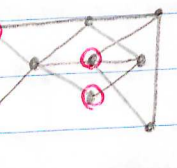
⇒ the hardest problem in NP is possibly hard ← compare the difficulties of problems

How to compare the difficulties of two problem?

IS (then it's easy) $\leq_m^P$ VC (suppose easy)

$x=(G,k)$ → $x=(G,n-k)$

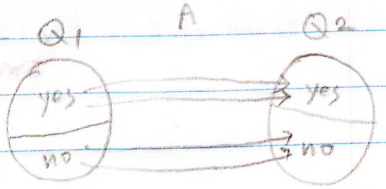
k, n-k



Def Q_1, Q_2 : decision problems

we say Q_1 is poly-time reducible to Q_2, written $Q_1 \leq_m^P Q_2$, if there is a poly-time algo A such that ← many reduction

Q_1		Q_2
x_1	$\xrightarrow{A}$	x_2
yes	iff	yes



NP : problems whose solutions can be verified easily

Q is in NP if there is an alg. A such that:

* for each yes-instance x, there is a y such that $A(x,y)=1$
* for each no-instance x, for all y, $A(x,y)=0$
* $A(x,y)$ runs in time polynomial in $|x|$

$Q_1 \leq_m^P Q_2$ if there is a poly-time alg A such that x is yes for Q_1, if and only if $A(x)$ is yes for Q_2.

$Q_1 \leq_m^P Q_2$

$x \xrightarrow[A]{} A(x)$

$IS \leq_m^P VC$

$(G,k) \rightarrow (G, n-k)$

Lemma if $Q_1 \leq_m^P Q_2$ and if Q_2 is in P, then so is Q_1.

pf since $Q_1 \leq_m^P Q_2$

$x \xrightarrow[\text{time}=O(|x|^c)]{A_1} A_1(x)$ (algorithm, instance) $\quad |A_1(x)| \leq O(|x|^c)$

$\downarrow A_2 \quad \text{Time} = O(|A_1(x)|^d) = O(|x|^{cd})$

$A_2(A_1(x)) = \text{yes/no}$

Q_1-solver(x)

1. $x_2 = A_1(x)$;
2. call $A_2(x_2)$
3. return "yes" if $A_2(x_2) = 1$

if $Q_1 \leq_m^P Q_2$

then

Q_1 is not harder than Q_2

Q_2 is not easier than Q_1

if Q_2 is easy, then so is Q_1

if Q_1 is hard, then so is Q_2

Def a problem Q is NP-hard if for all problem Q' in NP, $Q' \leq_m^P Q$

Verify

Since there are many problems in NP that cannot be solved easily, so an NP-hard problem seems very hard.

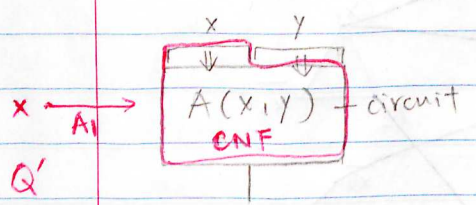
Def Q is NP-Complete if it is NP-hard and is in NP

Cook Theorem

SAT is NP-Complete

pf (roughly): Let $Q' \in$ NP, want $Q' \leq_m^P$ SAT

we have a verifier $A(x, y)$, x is an instance of Q'



$\Rightarrow Q' \leq_m^P$ SAT

Lemma: if $Q_1 \leq_m^p Q_2$ where Q_1 is NP-hard then so is Q_2.

pf $Q_1 \leq_m^p Q_2$

Q_1 is NP-hard

for all Q' in NP, $Q' \leq_m^p Q_1$

For any Q' in NP

$$Q' \leq_m^p Q_1 \leq_m^p Q_2$$

$$x' \xrightarrow[A_1]{} x_1 \xrightarrow[A_2]{} x_2$$

$\Rightarrow Q' \leq_m^p Q_2 \Rightarrow Q_2$ is NP-hard

SAT $\leq_m^p$ IS

$F = (\ )\overset{\text{and}}{\wedge}$ graph (G, K)

$(\ )\wedge$

$\vdots$

$(\ )\wedge$

$(\ )\wedge$

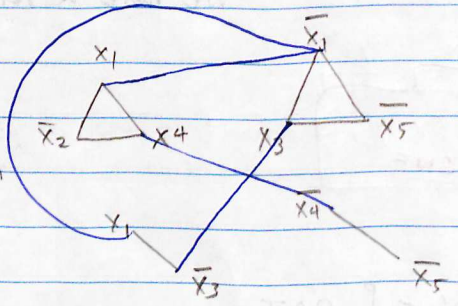
ex $(x_1 \vee \bar{x_2} \vee x_4) \wedge$

$(\bar{x_1} \vee x_3 \vee \bar{x_5}) \wedge$

$(x_1 \vee \bar{x_3}) \wedge$

$(\bar{x_4} \vee \bar{x_5})$

convert $\longrightarrow$ to graph



Dec-1-2015

* Q is NP-hard if for every problem Q' in NP, $Q' \leq_m^p Q$.
(NP-hard problems are hard)
* if $Q_1 \leq_m^p Q_2$ and $Q_2 \leq_m^p Q_3$, then $Q_1 \leq_m^p Q_3$. transitivity
* if $Q_1 \leq_m^p Q_2$ and Q_1 is NP-hard, then Q_2 is NP-hard.

SAT : given a CNF F, is F satisfiable ? NP-hard

IS : given G and K, is there an independent set of K vertices in G ?

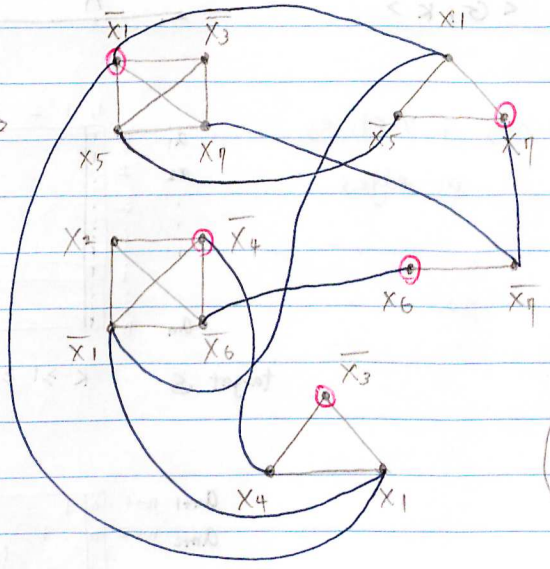
SAT $\leq_m^p$ IS

F $\xrightarrow{A}$ $\langle G_F, K_F \rangle$

$= C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$

$= (\bar{x}_1 \vee \bar{x}_3 \vee x_5 \vee x_7) \wedge$
$(x_1 \vee \bar{x}_5 \vee x_7) \wedge$
$(x_2 \vee \bar{x}_4 \vee \bar{x}_1 \vee \bar{x}_6) \wedge$
$(x_6 \vee \bar{x}_7) \wedge$
$(\bar{x}_3 \vee x_4 \vee x_1)$

$\Rightarrow$



$x_1 = 0$
$x_7 = 1$
$x_4 = 0$
$x_6 = 1$
$x_3 = 0$

yes $\Longleftrightarrow$ yes

NP-Complete problems :

SAT, IS, VC

Subset-Sum, Partition

$IS \leq_m^P VC$

$\langle G, k \rangle \longrightarrow \langle G, n-k \rangle$

Subset-Sum :

given n integers $a_1, \dots, a_n$ and S, can we pick some of them a_i's to make a sum of exact S ?

$VC \leq_m^P$ Subset-Sum

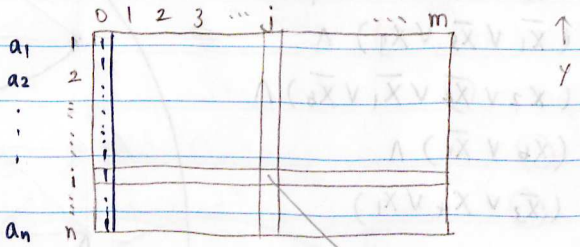
$\langle G, K \rangle \xrightarrow{A} \langle a_1, \dots, a_h ; S \rangle$

matrix $\langle a_1, a_2 \dots, a_{n+m} ; k \overbrace{2 2 \cdots 2}^{m} \rangle$

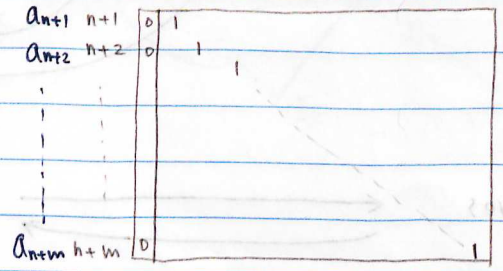
G has n vertices

m edges

↑ yes



target S k ≥1 ≥1 ≥1 ---------- 1 if e_j has V_i as an end



Partition :

given $a_1, \dots, a_n$, can you make them evenly split ? $\quad S = \sum_{i=1}^{n} a_i / 2$

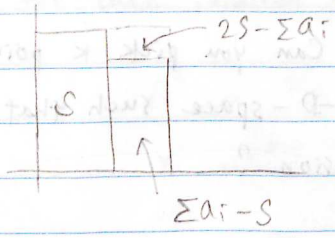
Partition $\leq_m^P$ Subset-Sum

$(a_1, \dots, a_n) \longrightarrow (a_1, \dots, a_n, S = \Sigma a_i / 2)$

Subset-Sum $\leq_m^P$ Partition

$(a_1, \dots, a_n, S) \longrightarrow (a_1, \dots, a_n, a_{n+1})$

$a_{n+1} = 2S - \Sigma a_i$



Dec-3-2015

NP-Complete problems

6 Basic NP-Complete problems

* Satisfiability (SAT): is a given CNF formula F satisfiable?
* Independent Set (IS): does a given graph G contain an independent set of K vertices?
* Vertex Cover (VC): does a graph G have a vertex cover of K vertices?
* Partition: can a given set of integer be partitioned evenly?

* Hamiltonian Path (HP): Does a given graph G contain a simple path that contains all vertices of G?

* 3-Dimension Matching (3DM): Can you pick K points from a given set S of n points in 3-D-space such that no two picked points agree on any dimension?
(remark: 2DM is in P)

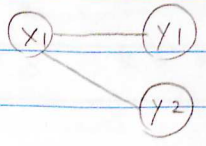
2DM

S: n points in 2D $\leq$ 2n real number

K:

(x_1, y_1)

(x_2, y_2)



partition $\leq_m^P$ Subset-sum

Max IS NP-hard

Min VC ratio ≤ 2

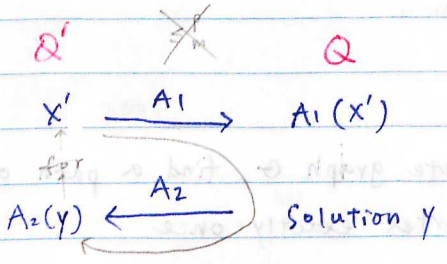
Max 3DM

Make span ratio ≤ 2

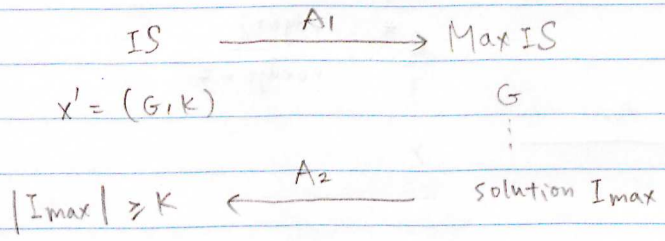
TSP

A problem Q (not necessarily in NP) is NP-hard if there is an NP-hard problem Q' and two poly-time algorithms A_1 and A_2 such that:

1. for each instance x' of Q', $A_1(x')$ is an instance x of Q
2. for each solution y of $A_1(x')$, $A_2(y)$ is a solution for x'

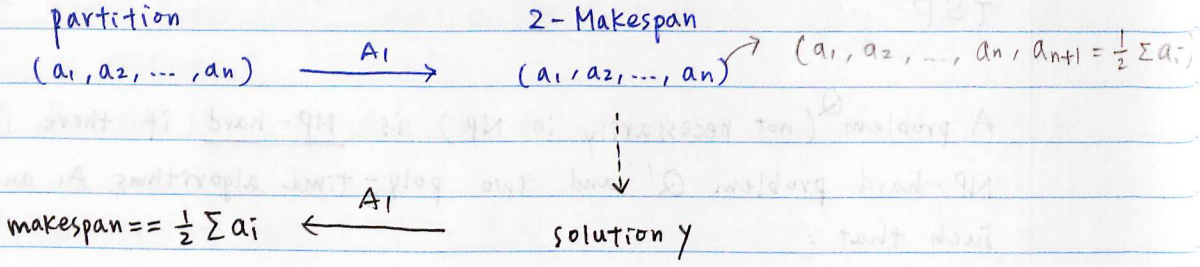


Show that Max IS is NP-hard:



2 - Makespan

given n jobs of times $t_1, t_2, \dots, t_n$, execute these jobs in two identical machines so finish ASAP

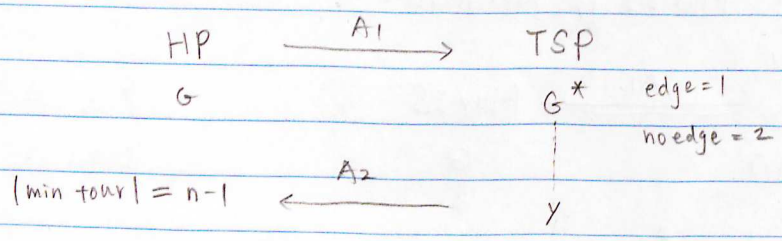


How about 3 - Makespan ?

TSP

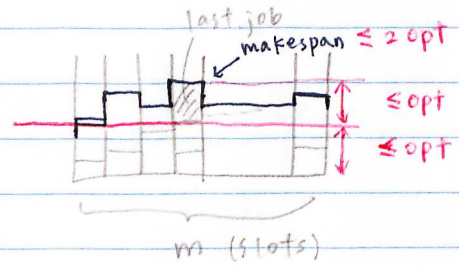
given a weighted complete graph G, find a path of min. weight that traverses each vertex exactly once

Show that TSP is NP-hard



MakeSpan

given n jobs $t_1, t_2, \dots, t_n$, maximize the makespan on m machines



quantity analysis

Graham Alg.

Min VC