Solution of Algorithms collection of problem.

1. Longest path in DAG.

solution: First run topology sort in $O(|V|)$ time.

if t is ahead of s, return "there is no path."

else drop other vertice before s and after t. then run DFS in $O(|V|+|E|)$ time.

return the path for s to t in sub Graph. ~~from~~ formed by DFS.

2. Finding the maximum ~~apea~~ polygon.

Solution. denote $MA[K,i]$ be the max k-gon area with k point selected from $1,2\cdots i$ and i is chosen. (the last vertice).

Then use DP:

$$MA[K,i] = \max_{j<i} \left\{ MA[k-1,j] + S(i, V_1, V_2) \;\middle|\; V_1, V_2 \in [1,j] \right\}$$

notice if $k < 3$ $MA = 0$.

Then we can change the start vertice from 1 to $2,3 \cdots n$.

So the time cost is $O(n \cdot n^2 \cdot m^2)$.

3. Longest palindrome subsequence.

Solution: denote $N[i,j]$ be the Length of longest palindrome sequence between $S[i]$, $S[j]$.

S is the whole sequence.

Then use DP:

$$N[i,j] = \begin{cases} N[i+1,j-1]+2, & \text{if } S[i]=S[j] \\ \max\{N[i+1,j], N[i,j-1]\} & \text{other wise.} \end{cases}$$

The result is $N[1, len(s)]$.

This algorithm can be done in $O(n^2)$.

4. Matrix-chain Multiplication.

Solution: denote $M[i,j]$ be the number of scalar Multiplication of $A_i \cdot A_{i+1} \cdots A_j$. $(i \le j)$.

Use DP:

$$M[i,j] = \begin{cases} 0, & \text{if } i=j \\ \min\limits_{i \le k < j} \{ M[i,k] + M[k+1,j] + p_{i-1} \times p_k \times p_j \}. & \text{otherwise.} \end{cases}$$

The solution is $M[1,n]$. time complexity $O(n^2)$.

5. Viterbi algorithm.

Solution: (a) denote $F(v_i, \sigma_m)$ be the feasibility that there exist a path $\langle \sigma_1, \sigma_2, \cdots \sigma_m \rangle$ in Graph and edge $e(v_{i-1}, v_i) = \sigma_m$. If so $F=1$ else $F=0$.

Use DP

$$F(v_i, \sigma_m) = \begin{cases} 1, & \text{if } \exists v_j \in adj(v_i),\ e(v_j, v_i) = \sigma_m \text{ and } F(v_j, \sigma_{m-1}) = 1. \\ 0, & \text{otherwise.} \end{cases}$$

During the search process, we could record the ~~successor or~~ ~~ancestor~~ precessor of each vertice.
We can check ~~if there~~ if $F(v_*, \sigma_k) = 1$ exist, if so, we could track the path back.
else, return there is no path. time cost $O(k \cdot |V|)$.

(b). denote $P(v_i, \sigma_m)$ be the ~~possibility~~ probability that path $\langle \sigma_1, \cdots \sigma_m \rangle$ exist in G and $e(v_{i-1}, v_i) = \sigma_m$.

Then use DP:

$$P(v_i, \sigma_m) = \max_{v_j \in adj(v_i)} \left\{ P(v_j, \sigma_{m-1}) \cdot p(v_j, v_i) \;\middle|\; e(v_j, v_i) = \sigma_m \right\}.$$

also record the precessor of each vertice during the algorithm running.

Then, check if there exis $P(v_*, \sigma_k) > 0$, if so return the path of max $P(v_*, \sigma_k)$ by tracking back. else return no path. The time complexity is $O(k \cdot |V|)$.

6. denote. $N[i,j]$ be the number of operations needed to change $X[:i]$ to $Y[:j]$.

Then use DP:

$$N[i,j] = \begin{cases} N[i-1,j-1] & \text{if } X[i] = Y[j] \\ \text{Min}\{N[i-1,j]+1,\ N[i,j-1]+1,\ N[i-1,j-1]+1\} & \text{if } X[i] \neq Y[j]. \\ i, & \text{if } j=0 \\ j, & \text{if } i=0. \end{cases}$$

The solution is $N[\text{len}(X), \text{len}(Y)]$.

7. See ZCH's ~~sto~~ solution set.

8. recurrences.

(a). $T(n) = 2T(n/2) + n\log n$.

(b) $T(n) = 2T(\sqrt{n}) + \log n$.

solution: (a). $T(n) = \sum_{k=0}^{\log n} 2^k \frac{n}{2^k} \log\left(\frac{n}{2^k}\right) = \sum_{k=0}^{\log n} n(\log n - k) = n\sum_{i=0}^{\log n} i = \frac{n}{2}\log^2 n$

$= O(n\log^2 n)$.

(b). set $2^m = n$. $T(2^m) = 2T(2^{\frac{m}{2}}) + m$ set $T_1(m) = T(2^m)$

so $T_1(m) = 2T_1(\frac{m}{2}) + m$. use master Theorem $\rightarrow$ $T_1(m) = O(m\cdot\log m)$

$m = \log n \rightarrow T(n) = O(\log n \cdot \log\log n)$.

9. maximal common subsequence.

solution: denote $N[i,j]$ be the max common subsequence of $A[:i]$, $B[:j]$.

Then use DP:

$$N[i,j] = \begin{cases} N[i-1,j-1]+1 & \text{if } A[i] = B[j] \\ \text{MAX}\{N[i-1,j],\ N[i,j-1]\} & \text{otherwise.} \end{cases}$$

Notice there are two loops for A and B respectively. So the time complexity is $O(m\cdot n)$.

When we update ith line of DP, we only need the information of i-1th line.

So the space complexity is $O(m+n)$.

10. Stick Game.

Solution.

denote win[i] be the label of whether the player will win when there are i sticks left.

Win[i] = 1 if he can win else Win[i] = 0.

Then use DP:

$$W[i] = \begin{cases} 1, & \text{if } W[i-1]=0 \text{ or } W[i-2]=0 \text{ or } W[i-3]=0 \text{ or } W[i-4]=0 \\ 0, & \text{if } W[i-1]=1 \text{ and } W[i-2]=1 \text{ and } W[i-3]=1 \text{ and } W[i-4]=1. \end{cases}$$

Time complexity is $O(n)$.

11. Monge Matrix.

solution: (a). suppose there exists n_1 and n_2, $f(n_1) > f(n_2)$.

$$\begin{array}{c} n_1 \\ n_2 \end{array}\begin{bmatrix} & & 0 \ f(n_1) \\ & 0 \ f(n_2) & \end{bmatrix}$$

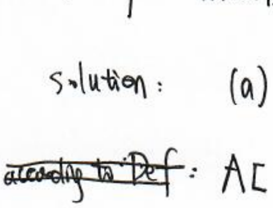
~~according to Def~~: $A[n_1, f(n_1)] + A[n_2, f(n_2)] \leqslant A[n_1, f(n_2)] + A[n_2, f(n_1)]$. ①

$A[n_1, f(n_2)] > A[n_1, f(n_2)]$ $A[n_2, f(n_1)] \geqslant A[n_2, f(n_2)]$ ②

① according to the Def of Monge matrix and ② according to the Def of $f(i)$.

① and ② are conflict. So $f(1) \leq f(2) \leq \cdots \leq f(m)$

(b). use Divided and conquer.



first we could calculate $f(\lfloor m/2 \rfloor)$ in $O(n)$ time.

Then according to the claim in (a). We can safely drop the up right & down left part of the maritx. Next calculate $f(\lfloor m/4 \rfloor)$ and $f(\lfloor 3\cdot m/4 \rfloor)$.

Then further drop other redudant block. ~~There The Matrix can be divided~~

Split matrix in $\underset{\downarrow}{\underline{\log m}}$ times, then take $O(n)$ to in each iteration. So calculate $f(n)$ takes $O(n \cdot \log m)$ time, use $O(m)$ to output result. $\rightarrow O(m + n\log m)$ #

$2^N = m \rightarrow N = \log m$.

12. Unit task scheduling.

solution.

$N_t(A)$ is the number of tasks in set A whose ddl is no later than t.

If $N_t(A) \leq t$, $\forall t$, A is an independent set. i.e. a set of tasks can be done with no penalty.

denote S be the set of all tasks, I to be the family of all independent set. It can be proved that $M = (S, I)$ is a matroid. (by prove two property: Hereditary / Exchange property).

Then use greedy Algorithm:

```
Greedy (S,I,P):     P: penalty (wi)
    A = { }
    sort S in p decreasing order.            time cost O(n²).
    for x in S:
        if Nt (A+x) ≤ t  for all t:
            A = A + {x}.
    return A
```

13. coin changing.

solution: By applying greedy algorithm, We have:

$$1 * C^k > \sum_{i=0}^{k-1} n_i \cdot C^i \quad ① \quad (n_i \text{ is the number of coin } c_i).$$

Suppose greedy algorithm fails to yield a optimal solution, then ① not ~~sat~~ consistent.

$\therefore$ $C^k \leq \sum_{i=1}^{k-1} n_i \cdot C^i$. By changing $(n_1, n_2 \cdots n_i)$ to one C^k, we could always reduce the total number of coins until ① satisfy. So greedy algorithm guarantee an optimal solution.

14. Schedule to minimize completion time.

Solution: We design a greedy algorithm. Start the tasks in a r_i increasing order.

Suppose current time is t_c and task a_i is running. If there exist a task a_j which is conflict with a_i ($r_j \leq t_c$), if $P_j < P_{i\,rest}$ (rest time of i to finish), then suspend ~~a_i~~ a_i, start a_j. denote a_i' ~~whose~~ with processing time $P_{i\,rest}$ as a "new task".

When a task is finished, pick task with smallest P_* and $r_* \leq t_c$ to start.

15. Suppose $|M^*|-|M|=m$, set P is defined as the set of edges which in M^* but not in M.

$$P=\{v_i\in V,\ \text{if } v_i \text{ in } M^* \text{ and } v_i \text{ not in } M\}.$$

M^*, M are matching so edges in P are not connected. Suppose edge in P can form n M-augmenting path with M, then these path are not connected either.

denote the length of these path as $2p_i+1$. Because pathes are non-connected, we can add one ~~path~~ edge to M by replacing a path, so $n=m$.

Note that each edge in M can only exist in one path, $|M|\le\sum_i p_i$ and the length of shortest path is $2t+t$, $\sum p_i \geq nt$.

$$\therefore \frac{|M^*|-|M|}{|M|}=\frac{m}{|M|}\le\frac{n}{\sum_i p_i}\le\frac{n}{nt}\le\frac{1}{t} \quad \therefore |M^*|\geq\frac{t}{t+1}|M^*|.$$

16. (probably not correct).

solution: denote $A(t_i,t_j)$ as the highest profit one can get ~~at~~ during time span $[t_i,t_j]$.

To maximum $A(0,t)$:

We firstly start jobs in r_i increasing order. When a job a_j is conflict with a current job, ($r_j\le t_{current}$), if the number of running jobs is less than k, then start a_j, else compare $p_i+A(c_i,c_j)$ with p_j, if p_j is larger, start a_j (cancel current job). else continue current job.

$$\text{Algorithm}\begin{cases}\text{start } j, & \text{if } N(\text{jobs})<k \text{ or } p_j>p_i+A(c_i,c_j)\\ \text{Continue } i, & \text{otherwise.}\end{cases}$$

when a job is finished, pick a_k ($a_k=$ ~~$r_k=t_c,\ p_k$~~ $\arg\max\{p_k \mid r_k=t_c\}$.

Total time cost $O(n^2)$.

17. Easy to check this problem is NP. Given a $G(V,E)$, and a solution X, we could delete X in G and then runing a Graph traversing algorithm to check whether G' contains a cycle.

To show it is NPC, we reduce it from vertex cover.

Let G, k be an instance of vertex cover. We produce a graph $G'(V',E')$ from $G(V,E)$ as follows.

for every vertex v in G, we creat v in G' as well. for an edge ~~#~~ $e(u,v)$ in G we creat a new vertex v' in G', and add edges (u,v'), (v',v), (v,u) in G'.

Now suppose G has a vertex cover of size k. (denote these vertex as S). We claim that S will be feed back set in G'. Indeed, any cycle in G' ~~much~~ must contail a pair (u,v) where (u,v) is an edge in G. Since S contains at least one of u, v, S must intersect this cycle. Thus after removing S from G', we will not have any cycle.

Conversely, consider a feedback set S' of size k in G'. First we claim that S' needn't contain any of the vertex v' — ~~if e(x,y)~~. because each cycle containing v' must also contain $e(v,v')$, so we can replace v' by v. Finally ~~an~~ a feed back set in S^* must contain a vertex from cycle ~~x~~ u, v, v' in G' where (u,v) in an edge in G. We just argue that S must contain either u or v. Thus, S' is a vertex cover in G.

18. Rearrangable Matrix.

(a).
$$\begin{bmatrix} \vdots & \\ \vdots & O \\ 0 & 1 \ 1 \cdots 1 \end{bmatrix}$$

(b). We can deduce this problem as finding a perfect matching in bipartite graph.

Denote vertex A be the row of matrix and B be column of the matrix. connected (a_i, b_j) if $m_{ij} = 1$. then form a Graph G.

Then we could run KM algorithm (for example) in $O(n^4)$ to find perfect matching of G. If perfect match exist, the Matrix M is rearrangeable.

19. Turan's bound.

solution: We firstly ~~ch~~ pick all the vertex in G one by one ~~into a random~~ to be a random sequence: $L = \{ v_1, v_2 \cdots v_n \}$. let $P(v_i)$ be the ~~sequence~~ position of v_i ~~at~~ in L.

Then we prove that $\sum_{v \in V} \frac{1}{deg(v)+1}$ is the size of an independant set, then claim the maximum set bigger than it.

We can define independent set as:
$$S = \{ v_i \in V : P(v_i) < P(v_j) \mid v_j \in adj(v_i) \}.$$

Intuitively, if V_i be chosen into the Independent set, its position in L must less than all its adjajcents.

So $E(V_i \in S) = \frac{1}{1+\deg(V_i)}$.

thus. $Len(S) = \sum_{v_i \in V} E(V_i) = \sum_{v_i \in V} \frac{1}{1+\deg(V_i)}$. thus $\alpha(G) \geq \sum_{v_i \in V} \frac{1}{1+\deg(v_i)}$

20. Multiple Interval scheduling.

Solution: Aparently, this problem is NP, since for a given job arrangement of size k, we can easily check whether it is feasible in a polynomial time.

Then We reduce the problem from Independent Set.

Given a set of jobs, we could form a graph G as follows:

each job is a vertex u in G. And the time intervals (eg: 1pm ~ 2pm) are edges: $e_1, e_2 \cdots e_n$. And edge e_i is connected to vertex u (u is endpoint) if the job u consist of time interval e_i $(t_{i-1} \sim t_i)$.

It is easy to check that G has an Independent set of size k only if there exist k jobs which could be a Multiple Interval scheduling instance and have no overlap. Because if two jobs are the endpoint of same edge, they cannot be taken simultanesly, thus by chose an Independent set, no jobs will conflict with eachother.

21. Solution: greedy algrithm

Suppose machine $1, 2 \cdots m$ are slow and $m+1, \cdots m+k$ are fast machine.

greedy algorithm:

for job sequence $\{1, 2 \cdots n\}$ assign job j to the machine with currently lowest workload. if there are lots of machine with same workload, choose one randomly. The time cost is $O(n)$.

denote t^* be optimal solution ~~of the~~ (~~minimum~~ max makespan).

thus $t^* \geq \frac{1}{2} \max_{i=1}^{n} t_i$.

Let T_i be the final workload of machine i.

Then $\sum_{i=1}^{n} t_i = \sum_{i=1}^{m} T_i + \sum_{i=m+1}^{m+k} 2T_i$

sum of job time

$\leq (2k+m)\, t^*$ ($T_i \leq t^*$ since t^* is the solution of maximum T_i).

Let T_r be the maximum workload. $T_r = \max_{i=1}^{m+k} T_i$ and t_j is the time for the last job j added to r. if r is a ~~slow~~ fast machine. before job j is added, its workload is $T_r - t_j$, according to our algorithm:

$$T_r - t_j \le T_i \quad \text{for } i \in [1, n] \quad \therefore (2k+m)(T_r - t_j) \le \sum_{i=1}^{m} T_i + \sum_{i=m+1}^{m+k} 2T_i = \sum_{i=1}^{n} t_i$$

$$\therefore T_r \le \frac{\sum_{i=1}^{n} t_i}{2k+m} + t_j \le t^* + 2t^* = 3t^* \quad \#$$

note if r is a ~~fast~~ slow machine the $T_r \le t^* + t^* = 2t^*$.

22. Densest K-subgraph.

solution:

Notice that only desicion problem can be a NP-complete. We first trans the problem to its desicion version: given a subset S of k vertice, Can the number of edges in $G(S)$ is larger than y?

We can easy to check ~~this prob~~ an answer satisfy the condition or not in polynomial time, thus this problem is NP.

The we reduce it from the known NPC problem CLIQUE:

problem CLIQUE:

Input: A undirect graph $G(V,E)$ and k.

output: Is there a clique of G containing exactly k vertices?

Reduction: G has a ~~vertice~~ CLIQUE of size k iff G has a subgraph of k vertice that Contains $k(k-1)/2$ edges.

prove: The CLIQUE of size k must have $k(k-1)/2$ edges. Conversely, if a SubGraph $S'(V,E)$ has $k(k-1)/2$ edges and k vertices. It must be a CLIQUE.

23. maximum converage.

(a). Firstly we change the ~~claim~~ problem to its decision version: Those k subsets of U, whether the total elements of their Union is larger than k?

Clearly, this is NP problem.

We do the reduction from know NPC problem Set cover.

The decision version of SC problem is: With k subsets chosen, whether the union.

of them can cover U?

reduction: By setting the number of elements $K = |U|$. We can see that the union of K ~~subg~~ subset cover U iff the totall elements of their union is $|U|$.

(b). design a greedy algorithm: given U and subsets $S_1, S_2 \cdots S_n$ where $\bigcup_{i=1}^{n} S_i = U$.

repeat:

- chose subset S_i with the maximum uncovered elements.
- set the elements in chose set as covered.

until K subsets are chosen.

Then we prove the algorithm is a $(1-\frac{1}{e})$ approximation for maximum coverage problem.

denote $\underline{OPT}$ be the optimal solution. $\underline{a_i}$ be the number of new covered elements at ith iteration; b_i is the total elements have been covered up to ith iteration (including ith iteration). thus $b_i = \sum_{j=0}^{i} a_i$; C_i is the number of elements which is uncovered after ith iteration. thus $b_i = OPT - C_i$. $(a_0 = b_0 = 0 \quad C_0 = OPT)$.

1° We firstly prove ~~$a_{i+1} \geq \frac{C_i}{K}$~~ $\boxed{a_{i+1} \geq \frac{C_i}{K}}$ which means the new covered element is larger than $\frac{1}{K}$ of uncovered element after ith iteration.

proof. Since : optimal solution covers all elements in K iteration. according to our algorithm, a_i is non-decreasing sequence. (since greedy). if $a_{i+1} < \frac{C_i}{K}$, the it is impossible. to cover all elements in K steps. #

2° Then we prove $C_{i+1} \leq (1-\frac{1}{K})^{i+1} OPT$

proof. When $i=0$, $C_1 \leq (1-\frac{1}{K}) OPT \qquad C_1 \leq OPT - \frac{1}{K} OPT$

$OPT - b_1 \leq OPT - \frac{1}{K} OPT \quad (C_i = OPT - b_i)$.

$b_1 \geq \frac{1}{K} OPT \quad (a_1 = b_1, C_0 = OPT)$

$a_1 \geq \frac{1}{K} C_0$

according to 1°, this is true.

Then we prove if $C_i \leq (1-\frac{1}{K})^i OPT$ is true the

$C_{i+1} \leq (1-\frac{1}{K})^{i+1} OPT$ is true. (归纳法).

$$C_{i+1} = C_i - a_{i+1}.$$

$$C_{i+1} \le C_i - \frac{C_i}{K} \quad (\text{according to } 1^\circ)$$

$$C_{i+1} \le (1-\tfrac{1}{K})\, C_i \le (1-\tfrac{1}{K}) \cdot (1-\tfrac{1}{K})^i \, OPT = (1-\tfrac{1}{K})^{i+1} \, OPT \ \#.$$

Then we show Our algorithm is $(1-\frac{1}{e})$ approximation of OPT.

$$OPT - C_{i+1} \ge OPT\left[1-(1-\tfrac{1}{K})^{i+1}\right]$$

$$b_{i+1} \ge OPT\left[1-\tfrac{1}{e}\right].$$ notice that b_i is the total element we chosen.

24. ploynomial multiplication.

Solution: denote $P_1(x) = \prod_{i=1}^{\lfloor n/2 \rfloor}(a_i x + b_i)$ $\quad P_2(x) = \prod_{i=\lfloor n/2 \rfloor+1}^{n}(a_i x + b_i)$.

~~Note~~ The time ~~cost~~ complexity of $P_1(x)$. $P_2(x)$ is the same as $\prod_{i=1}^{n}(a_i x + b_i)$.

Then we give a algorithm that can calculerte $P_1(x) \cdot P_2(x)$ in $O(n\log n)$.

Instead of multiple them directly (which $O(n^2)$), we use point-value mutiplication.

$$\begin{tikzcd}
\boxed{\begin{array}{l} a_0, a_1 \cdots a_n \\ b_0, b_1 \cdots b_n \end{array}} \arrow[d, "\begin{array}{l}\text{Calculate point value} \\ \text{of } P_1, P_2 \\ \text{in } O(n\log n)\end{array}", "\text{use FFT}"'] & \boxed{\begin{array}{l} C_0, C_1 \cdots C_n \\ \text{thus} \\ P(x) = C_0 + C_1 x + C_2 x^2 + \cdots \end{array}} \\
\boxed{\begin{array}{c} A(W_{2n}^0), B(W_{2n}^0) \\ \vdots \\ A(W_{2n}^{2n-1}), B(W_{2n}^{2n-1}) \end{array}} \arrow[r, "\text{point-multiply}", "\text{in } O(n)"'] & \boxed{\begin{array}{c} C(W_{2n}^0) \\ \vdots \\ C(W_{2n}^{2n-1}) \end{array}} \arrow[u, "\begin{array}{l}\text{interpolation (插值).} \\ \text{in } O(n\log n)\end{array}"']
\end{tikzcd}$$

total time cost is $O(n\log n)$.

25. Longest increasing subsequence.

Solution. denote $A[j]$ be the number of LIS in $S[:j]$ with last element j fixed.

Then Use DP:

$$A[j] = \begin{cases} \max\limits_{i<j} \{ A[i] + \text{sign}(S[i] < S[j]) \}. \end{cases}$$

Then final result is $\max\{A[i],\ i = 0,1,2\cdots \text{len}(s)\}$. total time cost: $O(n^2)$.

26.

(c) We firstly prove if n is not the power of 2. no ~~strategy~~ strategy could guarantee a win.

Suppose $n = r * 2^s$ with r ~~and~~ is odd and $r \geq 3$. Label cups $0, 1, 2 \cdots n-1$. Also 'face-up' and 'face-down' will be named '0' and '1'. Let the two special cups 0 and 2^s start out with oppsite orientations.

In the request move, consider the position with labels $i * 2^s$, $i = 0, 1 \cdots r-1$, since r is odd, the requests (move, non-move) among these r positions can not strictly alternate: there are two requests, separated by exactly 2^s which agree (either both are to move, or both are not to move). Image that the cups have been rotated before fulfilling this requests, So that our two cups (initially at 0 and 2^s) fall into these two positions. Then after the move, these two cups still have opposite orientations. This can go forever, nomatter what we request.

(b) 这道题 b.c 两问答案都看不懂. 估计不会考. 略去.

(a). $n=2$ 时. take three steps to win:

$(0,1)$, (0), $(0,1)$.

When $n=4$, 15 moves to win:

$(0,1,2,3) \to (0,2) \to (0,1,2,3) \to (0,1) \to (0,1,2,3) \to (0,2) \to (0,1,2,3) \to (0) \to$
$(0,1,2,3) \to (0,2) \to (0,1,2,3) \to (0,1) \to (0,1,2,3) \to (0,2) \to (0,1,2,3)$.

27. 5 stars. pass

28. We first show the graph is connected. Suppose G is not connected, ~~the it ...~~ then we can divided the component of G into two part V_0, V_1. since $|V| = |V_0| + |V_1|$. So V_i with smaller vertex has $|V_i| \leq \frac{|V|}{2} = \frac{n}{2} (i = 0 \text{ or } 1)$. consider a vertex in V_i, named V_j, $\deg(V_j) \geq \lceil n/2 \rceil$. $1 + \deg(V_i) > \frac{n}{2}$, which is conflict with. So G is connected.

We prove there is a Hamilton circuit by induction. Let P_m be the statement

" As long as $m+1 \leq n$, there is a path visiting $m+1$ distinct vertices with no repetitions."

P_0 is trivial — just a single vertex.

suppose P_m is true, then we have a path:

$$V_0 - V_1 - V_2 - \cdots - V_m.$$

we want to show that we can extend this to a circuit with one more element.

if V_0 or V_m has an adjacent node ~~that~~ not added in this path, then we can add it before V_0 (if $V_{new} \in adj(V_0)$) or after V_m (if $V_{new} \in adj(V_m)$).

but if all the neighbors of V_0 or V_m have already some where in this path. we could turn this path to a cycle.

suppose V_0 is not adjacent to ~~V_m~~ or we could connect (V_0, V_m)

$V_0 - V_1 - \cdots\ V_{t-1} - V_t - \cdots\cdots V_m$. then we break the link between V_{t-1} and V_t.

(we show we can always find (V_{t-1}, V_t) to break next).

and have the circuit: $V_t - V_{t+1} - \cdots - V_m - V_{t-1} - \cdots V_1 - V_0 - V_t$. ①

[We know that V_0 has $n/2$ neighbors, all of them are in this path and none are V_m. Let A be vertices adjacent to V_0. So $|A| \geq n/2$. Let B be vertices adjacent to V_m, So $|B| \geq n/2$

Let C be the set of vertices which are immediately after ~~some~~ vertex in B in this path.

Then $|C| = |B|$. If $A \cap C = \emptyset$, then $|A \cup C| \geq \frac{n}{2} + \frac{n}{2} \geq n$, so $A \cup C$ include all vertice. but $V_0 \notin A \cup C$, So $A \cap C \neq \emptyset$, thus there exist some vertex: $V_t \in A \cap C$ and $V_t \in A$ while $V_{t-1} \in B$]

If ~~$m+1 \leq n$~~ $m+1 = n$, then the path ① is a Hamiltion path (circuit) #

If $m+1 < n$, we can find a vertice w which is in G but not on path. and adjacent to V_u, then we can rotate our circuit ① So V_u is the first vetex and tack on w before it:

$w - V_u - V_{u+1} - \cdots V_m - V_{t-1} - \cdots V_1 - V_0 - V_t - \cdots V_{u-1}$.

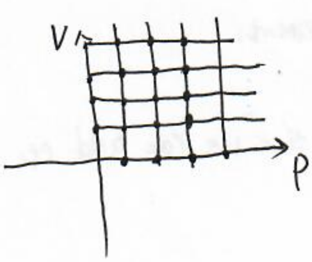
then by finding a new (V_{t-1}', V_t') to break, we could have a circuit of length $m+2$. (P_{m+1})

By induction, we know for every m, P_{m+1} is true. In particular, we could set $m = n-1$, then we have a circuit of length n.

29. solution. We can change this problem (find where the car is) to be find P_0 and V_0 of the car, because its position at time t is $P_0 + V_0 t$. and t is known.

suppose we check (P', V') at the t', then we could check $(-P', V')$, $(-P', -V')$, $(P', -V')$ at time $t'+1 \sim t'+3$, ~~so~~ then we note all these four states to be (P', V') and check all of them. successfully. (说检查(P',V')意思是在之后4t时间内将4种组合都进行检查).

Due to P_0, V_0 are integer, we could form a grid of (P, V), and check each point on the grid.



Check each point (p', V'), this can be done in finite time.

30. same as 19.

31.

a. We can run a greedy algorithm to form this path:

Connect two vertex with shortest path and ~~to~~ avoid form a sub cycle until a hamiltion path is formed.

(don't know how to prove)

b. according to cauchy's Inequality.

$$\left(|V_1V_2|+|V_2V_3|+\cdots|V_nV_1|\right)^2 \leq (1+1+\cdots1)\left(|V_1V_2|^2+|V_2V_3|^2+\cdots|V_nV_1|^2\right)$$
$$\leq n\cdot 4$$

$$\therefore\ |V_1V_2|+\cdots|V_nV_1| \leq 2\sqrt{n}.$$

32.

(a). This is obvious. We can run a greedy algorithm to color graph G:

For each uncolored vertex, we draw it and its neighbors ~~at~~ with different colors, since there are at most $1+\Delta$ ~~vertexe~~ vertices to draw, graph G can be colored in $\Delta+1$ colors.

(b). We show that In 3-Colorable graph $G(V,E)$, for each vertex $V \in G$, its neighbor $N(v)$ is a bipartite graph thus is 2-colorable.

proof. assume $N(V)$ is not bipartite, ~~thus~~ thus there will be an odd cycle with each vertex in the cycle connected to V. (shown in fig 32).

Fig. 32

obviously, this subgraph cannot be colored in 3 color since each of the node connect to three other nodes. #

~~EEF~~. We could use a greedy algorithm to finish this procedure. foreach uncolored node, use C_1 to draw it and C_2, C_3 to draw its neighbors.

(c). We run our algorithm in two steps.

(1). first partition the vertices with more than $\sqrt{n}$ degree:

$$S = \{ v \in V,\ deg(v) \geq \sqrt{n} \} \quad (n = |V|).$$

according to claim (b), we could use 3 color to draw S.

for each $v \in S$, use C_1 draw v and use C_1, C_2 draw its neighbors.

(2). For the rest of vertices, $V-S$. Use exactly $\sqrt{n}$ colors, since the max degree of $v \in V-S$ is $\sqrt{n}-1$. (according to (a), this is feasible).

clearly this algorithm runs in polynomial time.

Note that there are at most $\sqrt{n}$ vertices in S (due to the total number of vertices is n).

we use 3 color to draw S and $\sqrt{n}$ to draw $V-S$. thus $O(\sqrt{n})$-colors totally.

33. 5 stars. pass.

34. also called Coupon collector's problem.

solution: denote N_i the number of balls taken to fill i bins when $i-1$ bins have ~~been~~ been filled.

thus $N = N_1 + N_2 + \cdots N_n$.

We can find N_i is geometrically distributed with parapeter $p = \frac{n-(i-1)}{n}$

thus $E(N_i) = \frac{1}{p} = \frac{n}{n-i+1}$

thus $E(N) = \sum_{i=1}^{n} E(N_i) = n \sum_{i=1}^{n} \frac{1}{n-i+1}$ Set $t = n-i+1$

$\longrightarrow E(N) = n \sum_{t=1}^{n} \frac{1}{t}$ #

35.

We can solve this problem by using Min-cut - MAXflow algorithm by adding vertex s, s', t, t'. see fig 35.

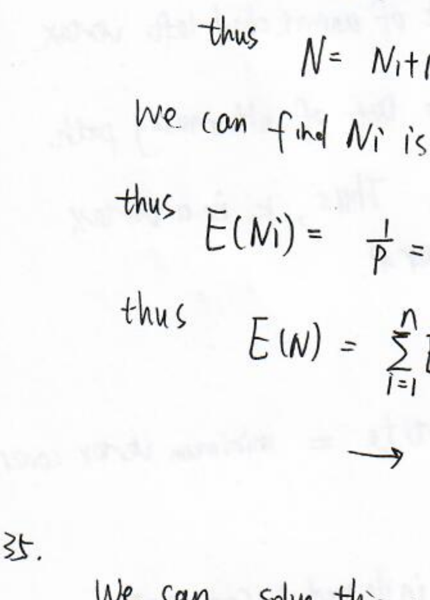


fig 35.

We connect s and s' then render the capability of $e(s,s')$ to be K.

Next We connect s' to all the vertices in one part of bipartite graph. and render the capability of $e(s', v_i)\ (v_i \in A)$ to be 1. And t' and t are created in same way. The cost of new added edges is 0.

Finally, run mincut-MAX-folw algorithm in polynomial time, we can get exactly $\langle |S| \rangle \rightarrow K$ Matches.

36.

Solution:

(a). König's theorem.

We show how to construct a minimum vertex cover from a maximum matching.

Let U be the set of unmatched vertices in L (Left side of bipartite, possibliy empty). and let Z be the set of vertices that are either in U ~~nor~~ or are connected to U by alternating path (path that alternate between edges that are in the matching and the edges that not in the matching). Let $K = (L \setminus Z) \cup (R \cap Z)$.

We show that K is a vertex cover. Each edge in G must either belongs to an alternating path so that its right endpoint in K, or it has a left endpoint in K. (If e is matched but not in alternating path, the left point of e belongs to $L \setminus Z$; if e is unmatched but not in alternating path, clearly its left endpoint cannot be in alternating path ~~in case~~ otherwise we could add e into alternating path. thus e's left point belongs to $L \setminus Z$).

Additionally, each vertex in K is an endpoint of matched edge.

For the vertex in $L \setminus Z$. is matched since Z is a superset of unmatched left vertex. And each vertex in $R \cap Z$ must be matched according to the Def of alternating path. However no matched point can have both its endpoints in K. Thus, K is a vertex cover of cardinality to M, and must be a minimum vertex cover #.

So $|K| = |M|$.

(b). We have shown in (a) that maximum matching in bipartite = minimum vertex cover. equivalent

To prove (b), We translate the claim to ~~equivalent~~ version:

In a bipartite graph $G(V,E)$, S ~~A~~ is a maximum independent set iff $V-S$ is a minimum vertex cover.

proof: "$\Longrightarrow$" suppose $V-S$ is not minimum vertex cover, there exis an edge e that both endpoints of $\in S$. obviously this is conflict with S is an independent set. thus $\Rightarrow$ is true

"$\Longleftarrow$" $\because$ $V-S$ is a minimum vertex cover, thus for each edge $e \in E$, at least one endpoint of e in $V-S$, thus at most one endpoint in S. So ~~bot~~ all the vertex in S are not connected #.

37. (only find the solution of finding a minimum number of edges, not weights).

This algorithm holds when all the weights are same.

Firstly we run maximum matching on G in ploynomial time.

Then run a greedy algorithm that for each unmatched vertex X_i, connect ~~it~~ it with X_j:

$$\arg\min_{X_j} \{ W(X_i, X_j) \mid X_j \in adj(X_i) \}.$$

then add $e(X_i, X_j)$ to E' until all vertex are incident with E'.

38. don't know.

39. (a) denote W: wolf G: goad C: cabbage H: human.

The solution is:

	left		right
①	W, C	$\xrightarrow{H,S}$	
②	W, C	$\xleftarrow{H}$	S
③	W	$\xrightarrow{H,C}$	S
④	W	$\xleftarrow{H,S}$	C
⑤	S	$\xrightarrow{H,W}$	C
⑥	S	$\xleftarrow{H}$	W, C
⑦		$\xrightarrow{H,S}$	W, C
⑧		$\longrightarrow$	W, C, S, H.

(b) We first show how to convert this problem into a graph G when an instance is given.

Use $S(A, B)$ be a state that A set of objects are on the left bank and B on the right bank.

When given n, k we could calculate all the states ~~in $O(n^2)$~~ (the number of the state is $O(2^n)$).

Then we could further check which two states are transitionable. (can change to each other in one move).

For each $S(A, B)$ is a vertex in graph G.

and for each pair of S_1 and S_2, connect them if they are transitionable. This will cost $O((2^n)^2)$.

Then finally, set $S(N, O)$ be start vertex s and $S(O, N)$ be end point t.

Run dijkstra algorithm to find a path $s \to t$. This will cost $O((2^n)^2)$.

If the path exist, by traversing the path, we get the solution or return it is impossible.

The total time complexity is $O(4^n)$.

40.

digits	num.	scope.
1	9	1~9
2	90	10~99
3	900	100~999
4	9000	1000~9999
5	90000	10000~99999
6	900000	100000~999999
7	1	
~~8~~		1000000

$\longrightarrow$ $N = 7\times1 + 6\times900000 + 5\times90000 + 4\times9000 + 3\times900 + 2\times90 + 1\times9 = 5\cancel{8}896.$

↑错写成了数字位数，应该求数字和.

Solution:

we divide $1\sim1000000$ in to 500000 pairs + 1 number:

$(1, 999998), (2, 999997), \cdots (499999, 500000), (999999, 0), 1000000.$

note that in first 500000 pairs, the sum of ~~digi~~ digits are same:

$$6\times9=54.$$

and the sum of last number's digits is 1.

thus: the total value is $N = 54\times500000+1 = 2700001$

41. Rumor Spreading.

Solution: We claim an algorithm that the total times of conversations ~~are less th~~ are equal to $2n-4$. (n: the number of the people).

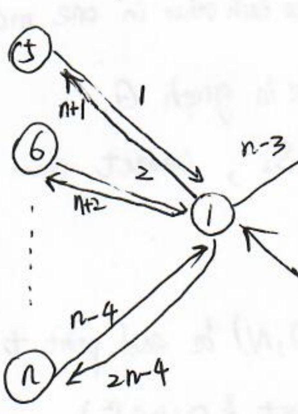


fig 41.

The change sequence are shown in fig41.

people $5\sim n$ firstly change with 1 sequentially. After that, 1 knows all the rumor from $5\sim n$.

Then change (1,2) and (3,4), after that 2 knows all the rumor besides 3 & 4.

by next change (1,3), (2,4), 1~4 knows all the rumors.

Finally, rechange 1 to $5\sim n$, to guarantee all people know all rumors.