算法设计与分析

动态规划

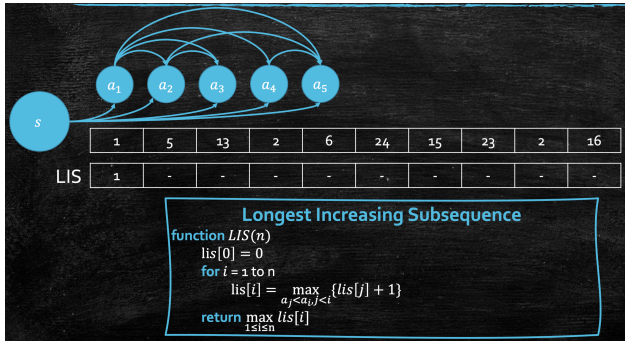
Guideline

1.Find subproblems.

2.Check whether we are in a DAG and find the topological order of this DAG. (usually, by hand.)

3.Solve and store the subproblems by the topological order.

最长上升子序列 (LIP)



编辑距离

Allowed operations:

- Insertion: insert a character to a specific location.

- Deletion: delete a character from a specific location.

- Replacement: rewrite a character at a specific location.

Another view:

- Alignment: Insert space with 0 cost.

- Insertion: rewrite a character from a space at a specific location.

- Deletion: rewrite a character to a space at a specific location.

- Replacement: rewrite a character at a specific location.

ED[i, j] : The edit distance between $X[1..i]$ and $Y[1..j]$. $ED[n, m]$: The edit distance between X and Y.

To solve $ED[i, j]$, consider the min of three cases

$-ED[i-1, j-1] + 1_{x_i \neq y_j}$

$-ED[i, j-1] + 1$

$-ED[i-1, j] + 1$

Running Time: $O(nm) \cdot O(1)$

背包问题

Input: n items with cost c_i and value v_i and a capacity W.

Output: Select a subset of items, with total cost at most W. The goal is to maximize the total value.

$f[i, w]$: the maximum value we can get by using the first i items, and with w budget.

Two options for item i

- Buy it: We can at most use $w - c_i$ budget before i.

- Not Buy it: We can at most use w budget before i.

- Solve $f[i, w] = \max\{f[i-1, w], f[i-1, w-c_i] + v_i\}$

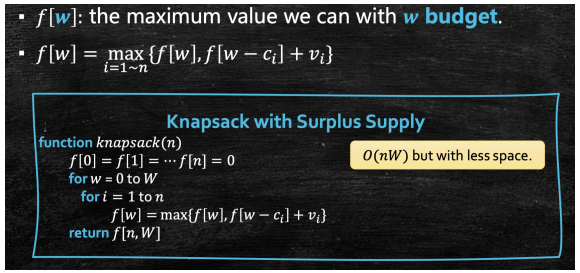
Running Time: $O(nW)$

上面这个是一个物品只能买一次的情况，如果一个物品可以反复购买的话，要做修改：New transfer

- Not Buy it anymore: We can at most use w budget before i.

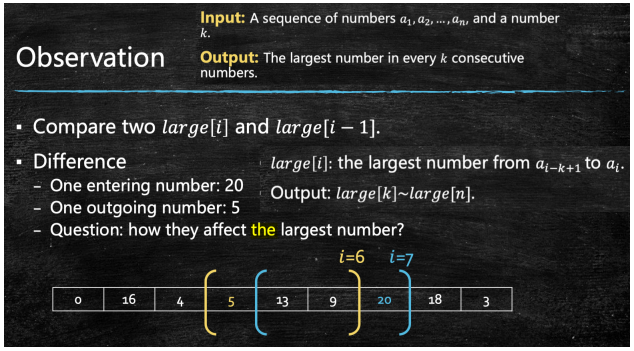
- Buy it: We can at most use $w - c_i$ budget before i and i.

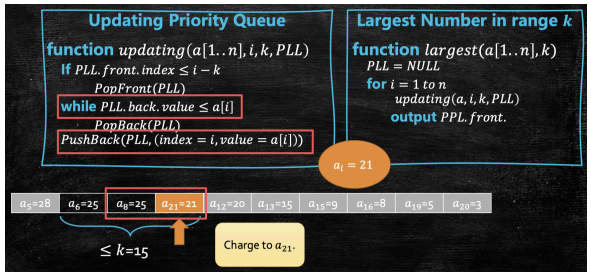
- Solve $f[i, w] = \max\{f[i-1, w], f[i, w-c_i] + v_i\}$.



$O(nW)$ is not polynomial! - Input: n items with cost c_i and value v_i, and a capacity W. - Input size: the unit of bits to represent the input. - $W = 2^N$ by using N bits - time complexity becomes $O\left(2^N\right)$.

Largest Number in k Consecutive Numbers



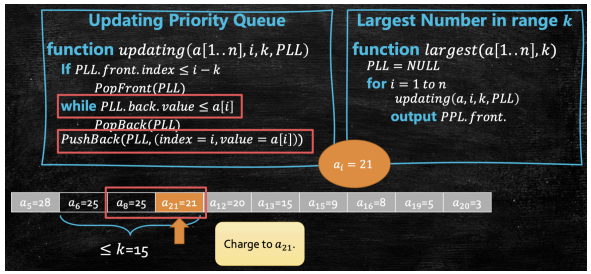


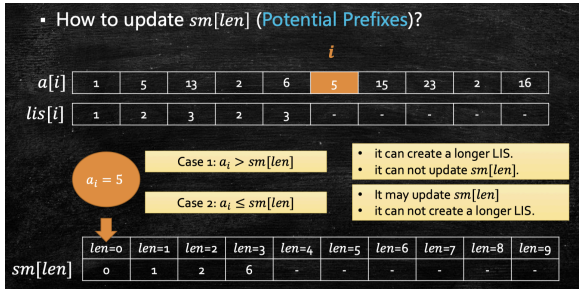
- The cost of n times updating has been charged to numbers.

Each number - Charged once when it is popped out. - Charged once when it is pushed in.

Totally: $O(n)$.

Revisit LIP





Initialize sm[0] $= 0$ For $i = 1$ to n

- Update $sm[len]$ by a_i

- It requires $\boldsymbol{O}(\max\{\boldsymbol{len}\} = i)$, 如果用二分优化，可以达到 $O(\log(\text{len})) = O(\log n)$

- Remark: now we do not kick everything we pass.

Output the largest len such that sm[len] $\neq$ " $-$ ".

Minimizing Manufacturing Cost

Input: A sequence of items with cost $a_1, a_2, \ldots, a_n$

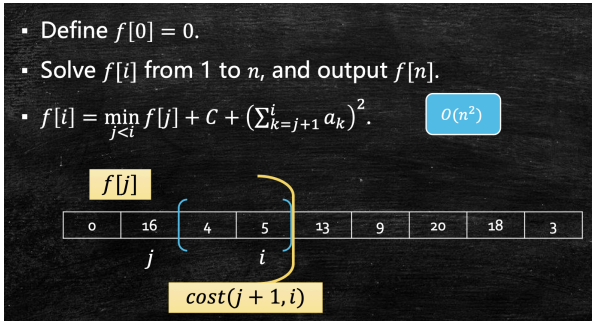
Need to Do:

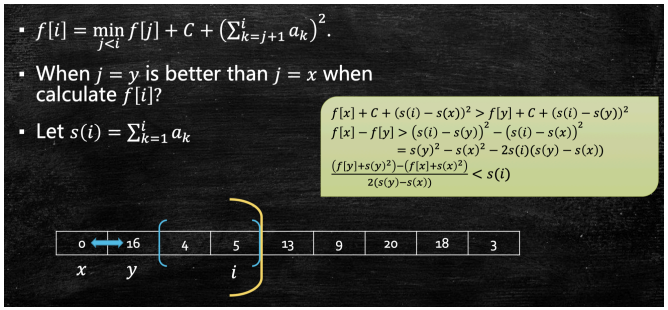
- Manufacture these items.

- Operation man(l, r) : manufacture the items from l to r.

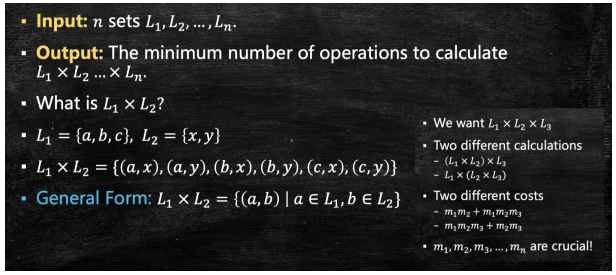
$-$ cost$(l, r) = c + \left(\sum_{i=l}^{r} a_i\right)^2$.

Output: The minimum cost to manufacture all items.





Product of Sets



Transfer: $c(i, j) = m_i m_{i+1} \ldots m_j + \min_{i \le k < j}\{c(i, k) + c(k + 1, j)\}$

n^2 subproblems, we use n iterations to calculate each. $O\left(n^3\right)$

All Pair Shortest Path

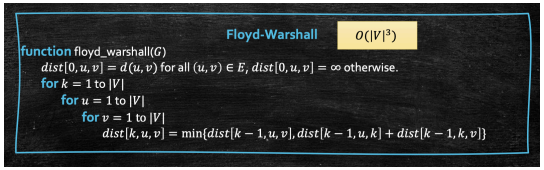
Input: A directed graph $G(V, E)$, and a weighted function $d(u, v)$ for all $(u, v) \in E$.

Output: Distance dist(u, v), for all vertex pair u, v.

Natural Generalization: dist $[k, u, v]$: the shortest distance from u to v among all k-edge-path (path with at most k edges).

Floyd-Warshall: dist $[k, u, v]$: the shortest distance from u to v that only across inter-vertices in $\{v_1 \ldots v_k\}$.

Remark: - We can label vertices from 1 to $|V|$. - dist $[0, u, v]$ is exactly $d(u, v)$ or ∞. (allow 0 inter-vertex) - dist $[|V|, u, v]$ is exactly what we want!



Traveling Salesman Problem (TSP)

Input: A complete weighted undirected graph G, such that $d(u, v) > 0$ for each pair $u, v (u \neq v)$.

Output: the cycle of n vertices with the minimum weight.(一个点只能走一次)

$$f[S, v] = \min_{k \in V} f[S - \{k\}, k] + d(k, v)$$

Time complexity ,$(n = |V|)$

$O\left(n2^n\right)$ subproblems, $O(n)$ solving.

- $O\left(n^2 2^n\right)$ totally!

Comparing to Brute-force - Brute-force: $\boldsymbol{O(n!)}$

Maximize Independent Set on Trees

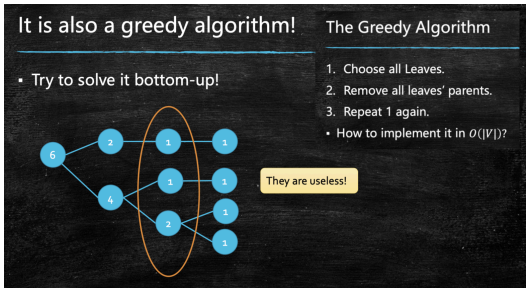
Input: an undirected tree $G = (V, E)$.

Output: an independent set with maximum cardinality (number of vertices)

$$f[v] = \max\left\{\sum_{u \in \text{ children } (v)} f[u], \sum_{u \in \text{ grandchildren } (v)} f[u] + 1\right\}$$

Looks $O\left(n^2\right)$. - We have n subproblems. - Each take $O(n)$ times. But it is $O(n)$.

- Each of its children and its grandchildren cost one. - On other words, each vertex only need to pay one for its parent and one for its grandparent.



具体实现只要在树上跑一遍 DFS，然后把碰到的叶子节点保留，回退时把父亲节点标记。

网络流问题

Definition 1. 网络流问题：在给定每条路的容量大小时，如何规划，让流入终点的东西最多。

Definition 2. *A Flow* $f : E \to \mathbb{R}_{\geq 0}, f(e)$ *for all* $e \in E$.

Definition 3. *Capacity Constraint: for each* $e \in E, f(e) \leq c(e)$.

Definition 4. *Flow Conservation: for each* $u \in V \backslash \{s, t\}$, $\Sigma_{v:(u,v)\in E} f(v, u) = \sum_{w:(u,w)\in E} f(u, w)$.

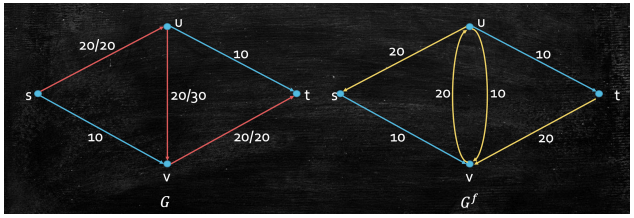
Definition 5. *Total flow:* $v(f) = \sum_{v:(s,v)\in E} f(s, v)$.

基于贪心的尝试

随变从图中选一条从 s-t 的路，然后塞满，接着继续找。

- 显然这种方法是有问题的，我们走着走着会发现，不好的取法封死了最优路线。所以我们需要 Cancellation!

Residual Network



剩余网络记录了每条路上“走了多少流量”以及“还能走多少流量”两种信息。“走了多少流量”就对应着我能回退多少流量，这让 Cancellation 有了实现的可能。

Definition 6. *Given* $G = (V, E), c$, *and a flow* f $G^f = \left(V^f, E^f\right)$ *and the associated capacity* $c^f : E^f \to \mathbb{R}^+$ *are defined as follows:*

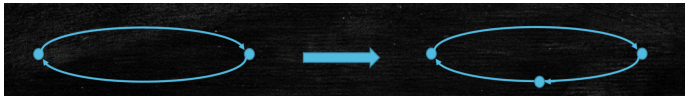
- $V^f = V$
- $(u,v) \in E^f$ *if one of the followings holds*
  - $(u,v) \in E$ *and* $f(u,v) < c(u,v)$ *: in this case,* $c^f(u,v) = c(u,v) - f(u,v)$
  - $(v,u) \in E$ *and* $f(v,u) > 0$ *: in this case,* $c^f(u,v) = f(v,u)$

Ford Fulkerson

Algorithm 1 Ford Fulkerson: $(G = (V, E), s, t, c)$

initialize f such that $\forall e \in E : f(e) = 0$; initialize $G^f \leftarrow G$;
while there is an s - t path p on G^f **do**
  find an edge $e \in p$ with minimum capacity
  for each $e = (u,v) \in p$ **(a small bug here) do**
    if $(u,v) \in E$: update $f(e) \leftarrow f(e) + b$
    If $(v,u) \in E$: update $f(e) \leftarrow f(e) - b$
  end for
  update G^f
end while
return f

在 for 循环中，如果原图中 u 和 v 之间有平行的两条反向边，那判断的时候会出问题。我们要如下调整图。



而如果有平行的两条同向边，直接合并就可以了。

Correctness

若所有的 Capacity 都是整数，我们有：

lemma 1. *Each while-loop iteration increase the value of f by at least 1.*

Theorem 1. *If each $c(e)$ is an integer, then the max flow f_{max} is an integer.*

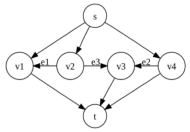
Proposition 1. *在整数和有理数情况下，Ford Fulkerson 一定会终止。*

如果不是整数，而是有理数，那也好办，我们可以通过给所有 Capacity 乘一个数让其都变为整数。然而，如果是无理数，则不一定:

Non-terminating example [edit]

Consider the flow network shown on the right, with source s, sink t, capacities of edges e_1, e_2 and e_3 respectively 1, $r = (\sqrt{5}-1)/2$ and 1 and the capacity of all other edges some integer $M \geq 2$. The constant r was chosen so, that $r^2 = 1 - r$. We use augmenting paths according to the following table, where $p_1 = \{s, v_4, v_3, v_2, v_1, t\}$, $p_2 = \{s, v_2, v_3, v_4, t\}$ and $p_3 = \{s, v_1, v_2, v_3, t\}$.

Step	Augmenting path	Sent flow	Residual capacities		
			e_1	e_2	e_3
0			$r^0 = 1$	r	1
1	$\{s, v_2, v_3, t\}$	1	r^0	r^1	0
2	p_1	r^1	r^2	0	r^1
3	p_2	r^1	r^2	r^1	0
4	p_1	r^2	0	r^3	r^2
5	p_3	r^2	r^3	r^3	0



Note that after step 1 as well as after step 5, the residual capacities of edges e_1, e_2 and e_3 are in the form r^n, r^{n+1} and 0, respectively, for some $n \in \mathbb{N}$. This means that we can use augmenting paths p_1, p_2, p_1 and p_3 infinitely many times and residual capacities of these edges will always be in the same form. Total flow in the network after step 5 is $1 + 2(r^1 + r^2)$. If we continue to use augmenting paths as above, the total flow converges to $1 + 2\sum_{i=1}^{\infty} r^i = 3 + 2r$. However, note that there is a flow of value $2M + 1$, by sending M units of flow along sv_1t, 1 unit of flow along sv_2v_3t, and M units of flow along sv_4t. Therefore, the algorithm never terminates and the flow does not even converge to the maximum flow.[4]

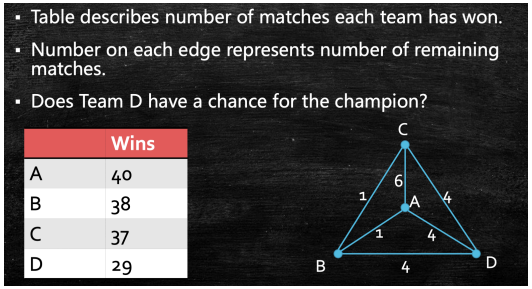
Another non-terminating example based on the Euclidean algorithm is given by Backman & Huynh (2018), where they also show that the worst case running-time of the Ford-Fulkerson algorithm on a network $G(V, E)$ in ordinal numbers is $\omega^{\omega^{|E|}}$.

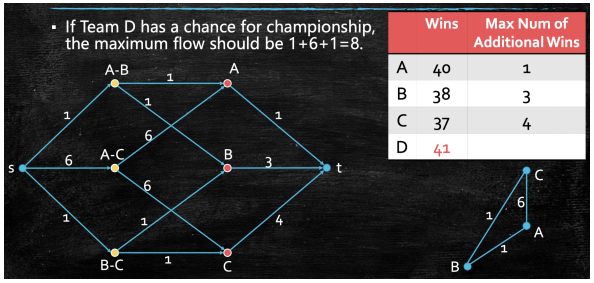
Time Complexity

WHILE 循环内部的操作都是 $O(E)$ 的，而我们由 Lemma 1 知道最多在 f_{max} 轮循环后，算法一定终止。所以复杂度为 $O(f_{max}|E|)$。

- 另一方面，我们可以直观地感受到，复杂度和 path 的选取很有关系，说明算法还有优化的空间。

一些应用



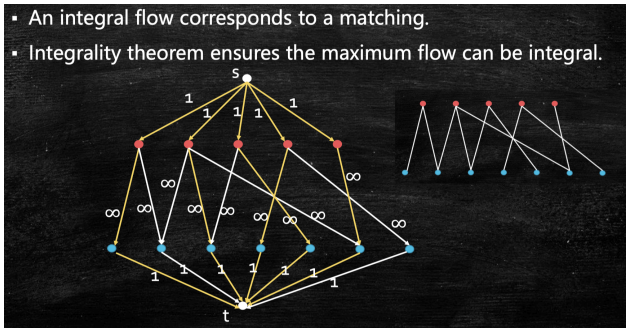


如果 D 队能赢，那就一定存在一个大小为 8 的最大流。如果存在一个大小为 8 的流，那我们就可以把比赛的结果看作一个整数流。

Maximum Bipartite Matching

定义. *Given a graph $G = (V, E)$, a matching M is a subset of edges that do not share vertices in common.*

- *The size of a matching is the number of edges in it.*
- *Input: A bipartite graph $G = (A, B, E)$.*
- *Output: A matching with the maximum size.*



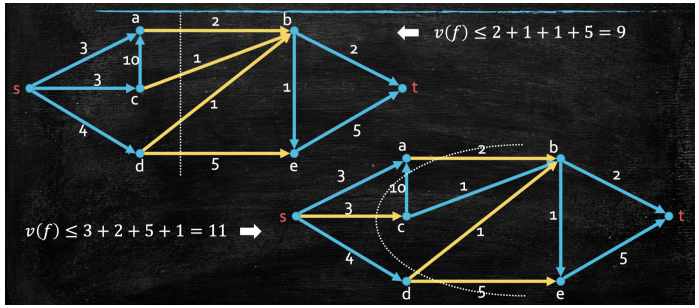
Dessert

- A graph is regular if all the vertices have the same degree.
- A matching is perfect if all the vertices are matched.
- Prove that a regular bipartite graph always has a perfect matching.

网络流正确性

定义. 在图论中，割是指将一个图分成两个非空的子图的边集。换句话说，割是一组边，删除这组边会导致图被分割成**至少两个连通分量**。最小割（*Min-cut*）通常是指具有最小权重的割（在无权图中为最少边数）。

直观来讲，割集给出了流的可能上界。



Minimum Cut Problem

定义. *s-t cut: Given weighted graph $G = (V, E, c)$ and $s, t \in V$, an $s-t$ cut is a partition of V to L, R such that $s \in L$ and $t \in R$. The value of the cut is defined by*

$$c(L,R) = \sum_{(u,v)\in E, u\in L, v\in R} c(u,v)$$

定义. *Min-Cut Problem: Given $G = (V, E, c)$ and $s, t \in V$, find the $s-t$ cut with the minimum value.*

Max-Flow-Min-Cut Theorem

Theorem 2. *Max-Flow-Min-Cut Theorem. The value of the maximum flow is exactly the value of the minimum cut:*

$$\max_f v(f) = \min_{L,R} c(L,R)$$

网络流: Running Time

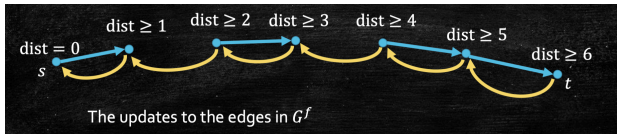
Algorithm 2 Edmonds-Karp Algorithm: $(G=(V,E), s, t, c)$

initialize f such that $\forall e \in E : f(e) = 0$; initialize $G^f \leftarrow G$
while there is an $s-t$ path on G^f **do**
  find such a path p by BFS
  find an edge $e \in p$ with minimum capacity b
  update f that pushes b units of flow along p;
  update G^f;
end while
return f

Why BFS?

- In the residual network G^f, a new appeared edge can only go from a vertex at distance $t+1$ to a vertex at distance t.
- Addition of such edges does not decrease the distance between s and u for every $u \in V$.

[**Key Observation**] $\text{dist}(u)$ in G^f is non-decreasing.



Weak Monotonicity to Strong Monotonicity

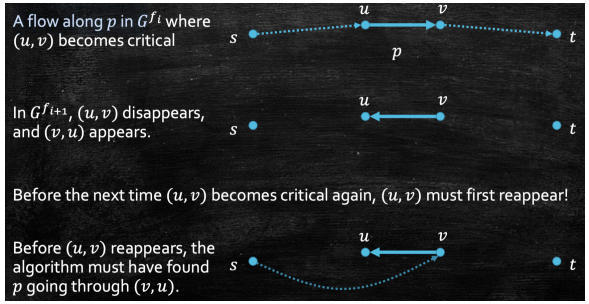
Key Observation 给了我们很好的性质，每次更新的时候每个点到 source 的距离不会减少。但是这还不够!

- Observation: At least one edge (u,v) on p is saturated, and this edge will be deleted in the next iteration.
- Each iteration will remove an edge from a vertex at distance i to a vertex at distance $i+1$.
- Intuitively, we cannot keep removing such edges while keeping the distances of all vertices unchanged.

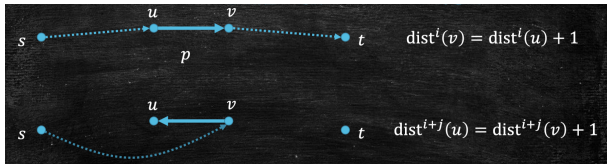
Suppose we are at the $(i+1)$-th iteration. f_i is the current flow, and p is the path found in G^{f_i} at the $(i+1)$-th iteration.

Definition 7. *Critical Edge: an edge (u,v) is critical if the amount of flow pushed along p is $c^{f_i}(u,v)$.*

A critical edge disappears in $G^{f_{i+1}}$, but it may reappear in the future. 我们来研究一条边两次成为 critical 之间发生了什么。



- Distance monotonicity: $\text{dist}^{i+j}(v) \geq \text{dist}^i(v)$.
- Thus, $\text{dist}^{i+j}(u) = \text{dist}^{i+j}(v) + 1 \geq \text{dist}^i(v) + 1 \geq \text{dist}^i(u) + 2$.
- The distance of u from s increases by 2 between two critical.



The distance of u from s increases by 2 between two "critical". Distance takes value from $\{0, 1, \ldots, |V|, \infty\}$, and never decrease. Thus, each edge can only be critical for $O(|V|)$ times. At least 1 edge becomes critical in one iteration. Total number of iterations is $O(|V| \cdot |E|)$. Each iteration takes $O(|E|)$ time. Overall time complexity for Edmonds-Karp: $O\left(|V| \cdot |E|^2\right)$.

It can handle the issue with irrational numbers!

Dinic's Algorithm

- Initialize f to be the empty flow and $G^f = G$.
- Iteratively do the followings until $\text{dist}(t) = \infty$:
  - Construct the level graph G_L^f for G^f.
  - Find a blocking flow on G_L^f.
  - Update f and G^f. (删除掉的边变成反边)

1. Build a level graph:

- Vertices at Level i are at distance i.
- Only edges go from a level to the next level are kept.**Backward edge** is not included!
- Can be done **in** $O(|E|)$ time using a similar idea to BFS.

2. Find a blocking flow on the level graph:

- Push flow on multiple $s-t$ paths.
- **Each $s-t$ path must contain a critical edge!**

How to find blocking flow?

Iteratively do the followings, until no path from s to t :

1. Run a frontward DFS.
2. Two possibilities:

- End up at t : in this case, we update f (by pushing flow along the path) and remove the critical edge
- End up at a dead-end, a vertex v with no out-going edges in G_L^f : in this case, we remove all the incoming edges of v

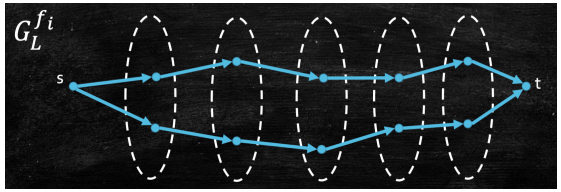
每次搜索后，至少有一条边被删除：$O(E)$; 每次搜索，最多有 $O(V)$ 步。BLOCKING FLOW 是一个流，不是一条路!

Proposition 2. *Time complexity for each iteration of Dinic's algorithm: $O(|V| \cdot |E|)$.*

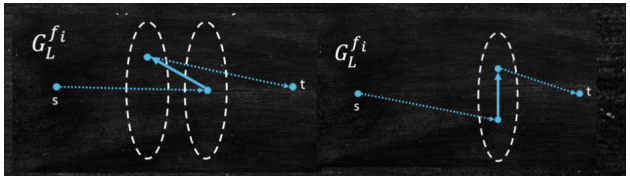
How many iterations before termination?

lemma 2. *In the level graph $G_L^{f_i}$ at every iteration i, every $s-t$ path has length $\text{dist}^i(t)$.*

lemma 3. *Every shortest $s-t$ path in G^{f_i} also appears in $G_L^{f_i}$.*



All the paths in $G_L^{f_i}$ with length $\text{dist}^i(t)$ are "blocked" after the i-th iteration. Thus, a path in the $(i+1)$-th iteration must use some edges that are not in $G_L^{f_i}$.If is a backward edge,dist(t) is increased by at least 2; if is not in $G_L^{f_i}$, then dist(t) increased by 1.



下面我们来严格证明: $dist^{i+1}(t) > dist^{i}(t)$

证明. Consider an arbitrary $s-t$ path p in $G_L^{f_{i+1}}$ with length $\text{dist}^{i+1}(t)$. We have $\text{dist}^{i+1}(t) \geq \text{dist}^{i}(t)$ by monotonicity. Suppose for the sake of contraction that $\text{dist}^{i+1}(t) = \text{dist}^{i}(t)$.

- Case 1: all edges in p also appear in $G_L^{f_i}$

Then p is a shortest path containing no critical edges in $G_L^{f_i}$
Contracting to the definition of blocking flow!

- Case 2: p contains an edge (u,v) that is not in $G_L^{f_i}$
  - If (u,v) was not in G^{f_i}, then (v,u) was critical in the last iteration(i-th iteration). We have $\text{dist}^{i}(u) = \text{dist}^{i}(v) + 1$.
  - If (u,v) was in G^{f_i} but not $G_L^{f_i}$, by the definition of level graph, we have $\text{dist}^{i}(u) \geq \text{dist}^{i}(v)$.

We have $\text{dist}^{i+1}(u) \geq \text{dist}^{i}(u)$ by monotonicity. In both cases above, $\text{dist}^{i}(u) \geq \text{dist}^{i}(v)$. □

We then prove $\text{dist}^{i+1}(v,t) \geq \text{dist}^{i}(v,t)$. 可以把这个看成是原来图的子图，起点是 v。然后根据 Key Observation. 即可。综上:

$$\begin{aligned}
\text{dist}^{i+1}(t) &= \text{dist}^{i+1}(u) + 1 + \text{dist}^{i+1}(v,t) \\
&\geq \text{dist}^{i}(u) + 1 + \text{dist}^{i}(v,t) && \text{(Fact ii and iii)} \\
&\geq \text{dist}^{i}(v) + 1 + \text{dist}^{i}(v,t) && \text{(Fact } i\text{)} \\
&\geq \text{dist}^{i}(t) + 1 && \text{(triangle inequality)}
\end{aligned}$$

于是我们发现，Dinic 算法的每一轮后，dist(t) 都会增加 1，那最多增加 $O(V)$ 次，也就说明 iteration 的次数最多是 $O(V)$, 从而总时间就是 $O(V^2E)$。

Hopcroft–Karp–Karzanov algorithm

Find a maximum bipartite matching in $O(|E| \cdot \sqrt{|V|})$ time.

- Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time. Iteratively do the followings, until no path from s to t :
  - Perform DFS from s
  - If we reach t, delete all edges on the s - t path (capacity is one!) and start over from s.
  - If we ever go backward, delete the edge just travelled. (capacity is one!)
- Step 2: Number of iterations is at most $2\sqrt{|V|}$. let f be the flow after $\sqrt{|V|}$ iterations.

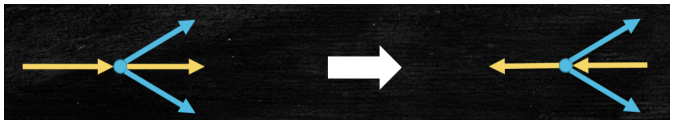
  Claim: the maximum flow in G^f has value at most $\sqrt{|V|}$. If the claim is true, we can stop after another $\sqrt{|V|}$ rounds.

Observation: In each iteration, for each $v \in V \backslash \{s,t\}$, either its in-degree is 1 , or its out-degree is 1 .

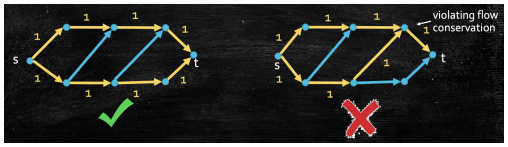
证明. At the beginning, this is clearly true.

- For each iteration, the amount of flow going through v is either 0 or 1.
- If it is 0, v's in-degree and out-degree are unchanged.
- Otherwise, exactly one in-edge and one out-edge are flipped; the property is still maintained.

□



By Integrality Theorem: there exists a maximum integral flow f' in G^f. f' consists of vertex-disjoint paths with flow 1 !

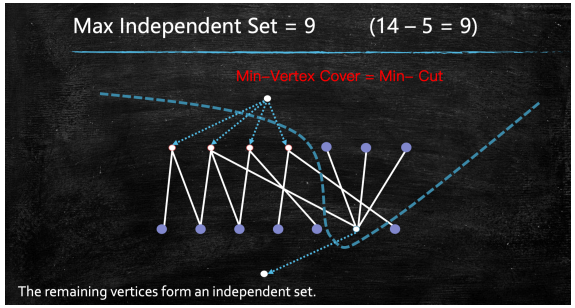


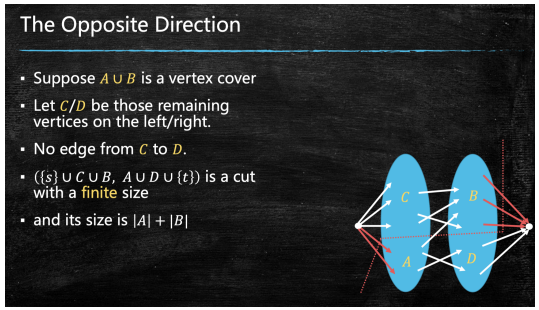
Max-flow on G^f, f', is integral and consists of edge-disjoint paths. By our analysis to Dinic's algorithm, $\text{dist}^{G^f}(s,t) \geq \sqrt{|V|}$. Each path in f' has length at least $\sqrt{|V|}$. There are at most $\frac{|V|}{\sqrt{|V|}} = \sqrt{|V|}$ paths in f' by vertex-disjointness.

$$v(f') \leq \sqrt{|V|}$$

Definition 8. *Independent Set*
Given an undirected graph $G = (V, E)$, a subset of vertices $S \subseteq V$ is an independent set if there is no edge between any two vertices in S.

Definition 9. *Vertex Cover*
Given an undirected graph $G = (V, E)$, a subset of vertices $S \subseteq V$ is a vertex cover if S contains at least one endpoint of every edge.





线性规划

标准形式:

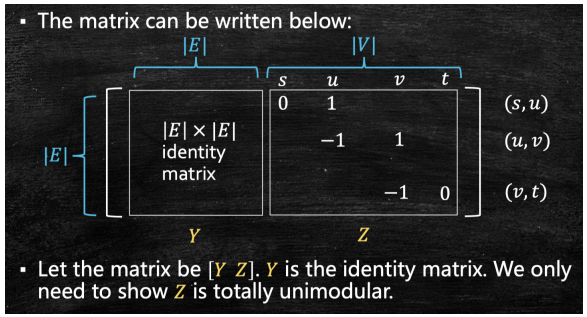
$$\begin{aligned}
\text{maximize} \quad & \mathbf{c}^\top \mathbf{x} \\
\text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\
& \mathbf{x} \geq \mathbf{0}
\end{aligned}$$

The maximum flow problem can be formulated by a linear program.

$$\begin{aligned}
\text{maximize} \quad & \textstyle\sum_{u:(s,u)\in E} f_{su} \\
\text{subject to} \quad & 0 \leq f_{uv} \leq c_{uv} && \forall (u,v) \in E \\
& \textstyle\sum_{v:(v,u)\in E} f_{vu} = \sum_{w:(v,w)\in E} f_{uw} && \forall u \in V \backslash \{s,t\}
\end{aligned}$$

Ford-Fulkerson Method implements the simplex method.
Dual Form:

$$\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{(u,v)\in E} c_{uv} y_{uv} \\
\text{subject to} \quad & y_{su} + z_u \geq 1 && \forall u : (s,u) \in E \\
& y_{vt} - z_v \geq 0 && \forall v : (v,t) \in E \\
& y_{uv} - z_u + z_v \geq 0 && \forall (u,v) \in E, u \neq s, v \neq t \\
& y_{uv} \geq 0 && \forall (u,v) \in E
\end{aligned}$$

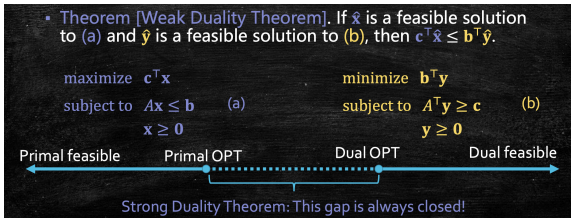


y_{uv} describes if edge (u,v) is cut:

$$y_{uv} = \begin{cases} 1 & \text{if } (u,v) \text{ is cut} \\ 0 & \text{otherwise} \end{cases}$$

z_u describes u 's "side":

$$z_u = \begin{cases} 1 & \text{if } u \text{ is on the } s\text{-side} \\ 0 & \text{if } u \text{ is on the } t\text{-side} \end{cases}$$



Definition 10. *A matrix A is totally unimodular if every square submatrix has determinant 0,1 or -1 .*

Theorem 3. *If $A \in \mathbb{R}^{m\times n}$ is totally unimodular and $\mathbf{b}$ is an integer vector, then the polytope $P = \{\mathbf{x} : A\mathbf{x} \le \mathbf{b}\}$ has integer vertices.*

Proof. If v $\in \mathbb{R}^n$ is a vertex of P. Then there exists an invertible square submatrix A' of A such that $A'v = b'$ for some sub-vector $\mathbf{b}'$ of $\mathbf{b}$. By Cramer's Rule, we have $v_i = \frac{\det(A'_i|\mathbf{b}')}{\det(A')}$, where $\left(A'_i \mid \mathbf{b}'\right)$ is the matrix A' with i-th column replaced by $\mathbf{b}'$. $\det(A') = \pm 1$ and $\det\left(A'_i \mid \mathbf{b}'\right) \in \mathbb{Z}$. Thus, $\mathbf{v}$ is integral.

If A is totally unimodular, then so are $A^\top, [I\ A], [A\ I], \begin{bmatrix} I \\ A \end{bmatrix}$, and $\begin{bmatrix} A \\ I \end{bmatrix}$. If any of $A^\top, [I\ A], [A\ I], \begin{bmatrix} I \\ A \end{bmatrix}$, and $\begin{bmatrix} A \\ I \end{bmatrix}$ is totally unimodular, then so is A.

Corollary. If A is totally unimodular, then the optimal solution to LP (a) is integral when $\mathbf{b}$ is integral, and the optimal solution to LP (b) is integral when $\mathbf{c}$ is integral.

maximize $\mathbf{c}^\top \mathbf{x}$
subject to $A\mathbf{x} \le \mathbf{b}$ (a)
$\mathbf{x} \ge \mathbf{0}$

minimize $\mathbf{b}^\top \mathbf{y}$
subject to $A^\top \mathbf{y} \ge \mathbf{c}$ (b)
$\mathbf{y} \ge \mathbf{0}$

Lemma 1. OPT dual is an upper-bound to min-cut. Proof. Let $(\mathbf{y}^*, \mathbf{z}^*)$ be an integral optimal solution. Let $C = \{(u,v) \in E : y^*_{uv} \ge 1\}$. We have shown removing C disconnect t from s. Let $L \subseteq V$ be the vertices reachable from s after removing C, and $R = V \backslash L$. Then $\{L, R\}$ is an $s-t$ cut. For min-cut $\{L^*, R^*\}$, we have $c(L^*, R^*) \le c(L, R)$

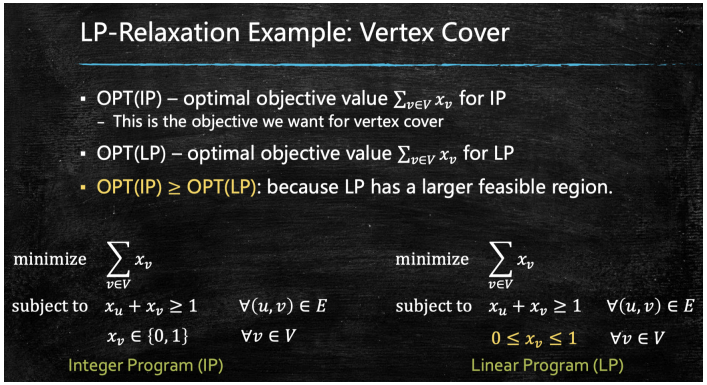
$$= \sum_{(u,v)\in E: u\in L, v\in R} c_{uv} \le \sum_{(u,v)\in E: u\in L, v\in R} c_{uv} y^*_{uv} = \mathrm{OPT}_{\mathrm{dual}}$$

Lemma 2. $\mathrm{OPT}_{\mathrm{dual}}$ is a lower-bound to min-cut. Proof. Let $\{L^*, R^*\}$ be a min-cut. We construct a LP solution:

$$y_{uv} = \begin{cases} 1 & \text{if } u \in L^*, v \in R^* \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad z_u = \begin{cases} 1 & \text{if } u \in L^* \\ 0 & \text{if } u \in R^* \end{cases}$$ It is easy to verify that the solution is feasible... Then,

$$\mathrm{OPT}_{\mathrm{dual}} \le \sum_{(u,v)\in E} c_{uv} y_{uv} = \sum_{(u,v)\in E: u\in L^*, v\in R^*} c_{uv} = c(L^*, R^*)$$

LP-Relaxation



P, NP, NP-Complete

Definition 11. *A Turing Machine $\mathcal{A}$ is a polynomial time TM if there exists a polynomial p such that $\mathcal{A}$ always terminates within $p(|x|)$ steps on input x.*

A decision problem $f : \Sigma^* \to \{0,1\}$ is in P, if there exists a polynomial time TM $\mathcal{A}$ such that $-\mathcal{A}$ accepts x if $f(x) = 1$ - $\mathcal{A}$ rejects x if $f(x) = 0$ Problems in **P** are those "easy" problems that can be solved in polynomial time.

P 问题

[PATH] Given a graph $G = (V, E)$ and $s, t \in V$, decide if there is a path from s to t. - Build a TM that runs BFS or DFS at s; accept if t is reached; reject if the search terminates without reaching t. - PATH $\in$ **P**

[k-FLOW] Given a directed graph $G = (V, E), s, t \in V$, a capacity function $c : E \to \mathbb{R}^+$, and $k \in \mathbb{R}^+$, decide if there is a flow with value at least k. - Build a TM that implements Edmonds-Karp, Dinic's, or other algorithms. - k-FLOW $\in$ **P**

[PRIME] Given $k \in \mathbb{Z}^+$encoded in binary string, decide if k is a prime number. - [Agrawal, Kayal & Saxena, 2004] PRIME $\in$ **P**

NP 问题

NP: Problems whose yes instances can be efficiently verified if hints are given.

Definition 12. *A decision problem $f : \Sigma^* \to \{0,1\}$ is in NP if there exist a polynomial q and a polynomial time TM $\mathcal{A}$ such that - If x is a yes instance $(f(x) = 1)$, there exists $y \in \Sigma^*$ with $|y| \le q(|x|)$ such that $\mathcal{A}$ accepts the input (x, y) - If x is a no instance $(f(x) = 0)$, for all $y \in \Sigma^*$ with $|y| \le q(|x|)$ such that $\mathcal{A}$ rejects the input (x, y)*

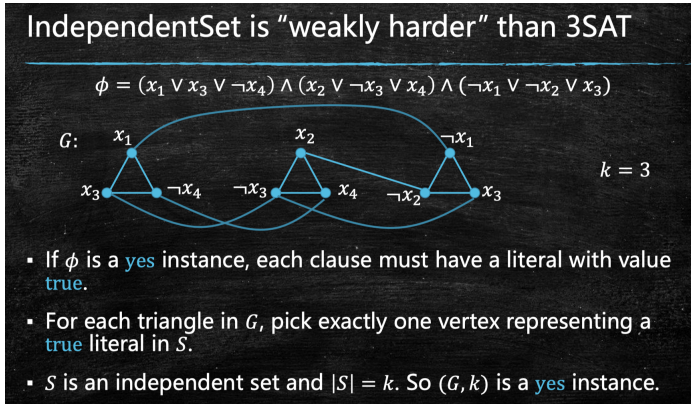
The string y is called a certificate. SAT, VertexCover, IndependentSet, SubsetSum, HamiltonianPath are all in NP.

Theorem 4. *$P \subseteq NP$*

- Proof. If a decision problem $f : \Sigma^* \to \{0,1\}$ is in P, we will show it is in NP. - By definition of P, there exists a polynomial time TM $\mathcal{A}$ such that $\mathcal{A}$ accepts x if and only if $f(x) = 1$. - Let $\mathcal{A}'$ be a TM such that it outputs $\mathcal{A}(x)$ on input (x, y). That is, $\mathcal{A}'$ implements $\mathcal{A}$ and ignore y. - If $f(x) = 1$, there exists y, say, $y = \emptyset$, such that $\mathcal{A}'$ accepts (x, y) - If $f(x) = 0$, for all y, $\mathcal{A}'$ rejects (x, y). - Thus, $f \in NP$.

Theorem 5. *3-SAT is weakly harder than SAT*

In general: - $(\ell_1 \vee \cdots \vee \ell_k) =$
$(\ell_1 \vee \ell_2 \vee y_1) \wedge (\neg y_1 \vee \ell_3 \vee y_2) \wedge \cdots \wedge (\neg y_{k-2} \vee \ell_{k-1} \vee \ell_k)$
- If a literal ℓ_i is true, we can make all RHS clauses true by properly setting y_i 's



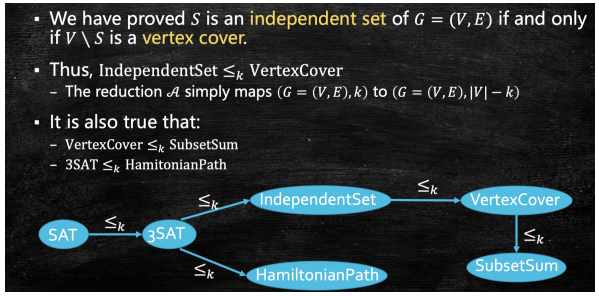
Reduction

A decision problem f Karp reduce to (or simply, reduce to) a decision problem g if there is a polynomial time TM $\mathcal{A}$ such that:

- $\mathcal{A}$ outputs a yes instance of g if a yes instance of f is input
- $\mathcal{A}$ outputs a no instance of g if a no instance of f is input

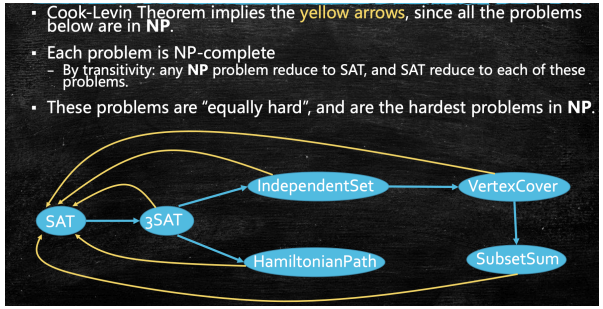
Denoted as $f \leq_k g$ - Very intuitive: the difficulty level of f is weakly less than that of g

We have just proved:1.SAT $\leq_k$ 3SAT 2.3SAT $\leq_k$ IndependentSet



Definition 13. *A decision problem f is NP-hard if $g \leq_k f$ for any problem $g \in NP$.*

Definition 14. *A decision problem f is NP-complete if $f \in$ **NP** and $g \leq_k f$ for any problem $g \in NP$.*



Theorem 6. *If f is NP-complete and $f \in$ **P**, then* **P** = **NP**.

Proof. Suppose there is a polynomial time TM $\mathcal{A}$ that decides f. We will show $g \in$ **P** for any $g \in$ **NP**.

Since f is NP-hard, $g \leq_k f$, and let $\mathcal{A}'$ be the polynomial time TM that does the reduction.

Then $\mathcal{A} \circ \mathcal{A}'$ is the polynomial time TM that decides g. Thus, $g \in$ **P**.

Theorem 7 (Ladner's Theorem)**.** *If $P \neq NP$, then there exist decision problems that are neither in P nor NP-complete. Such problems are called NP-intermediate. However, we do not know any "natural" NP-intermediate problems.*

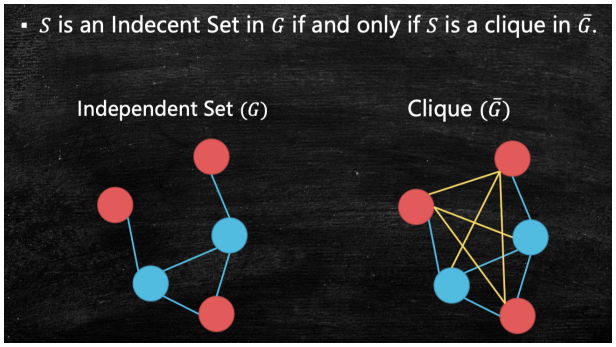
For decision problems: NP-complete = NP-hard + (in NP) **There are NP-hard problems that are not in NP; these problems are even harder than NP-complete problems.**

Independent Set $\leq_k$ Vertex Cover

When we receive an Independent Set Input (G, k). Just input $(G, |V| - k)$ to the Vertex Cover solver

- If G has a $|V| - k$ size vertex cover, we have $|S| = k$, where S is an independent set.
- If G has a size k independent set, we should have a $|V| - k$ size vertex cover.

Independent Set $\leq_k$ Clique



When we receive an Independent Set Input (G, k). Just input $(\bar{G}, k)$ to the Clique solver

- If G has a k size independent set, we have $|S| = k$, where S is a clique of $\bar{G}$.
- If $\bar{G}$ has a size k clique, we should have a k size independent set in G.

VertexCover $\leq_k$ SubsetSum

[VectorSubsetSum] Given a collection of integer vectors $S = \{\mathbf{a}_1, \ldots, \mathbf{a}_n : \mathbf{a}_i \in \mathbb{Z}^m\}$ and a vector $k \in \mathbb{Z}^m$, decide if there exists $T \subseteq S$ with $\sum_{\mathbf{a}_i \in T} \mathbf{a}_i = \mathbf{k}$

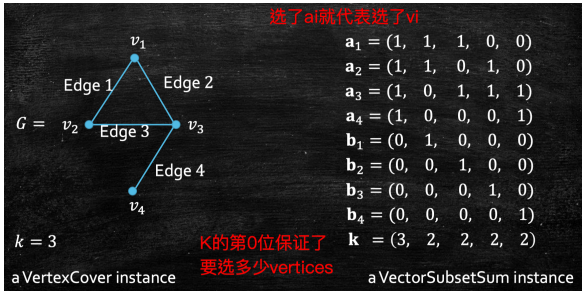
Given a VertexCover instance $(G = (V, E), k)$, we will construct a VectorSubsetSum instance $(S, \mathbf{k})$.

First, we label the edges with $1, 2, \ldots, |E|$ (in arbitrary order).

For each $v_i \in V$, construct a $(|E| + 1)$-dimensional vector $\mathbf{a}_i \in S$ such that $\mathbf{a}_i[0] = 1$ and for each $j = 1, \ldots, |E|$:

$$\mathbf{a}_i[j] = \begin{cases} 1 & \text{if } v_i \text{ is an endpoint of edge } j \\ 0 & \text{otherwise} \end{cases}$$

For each edge j, construct $\mathbf{b}_j \in S$ where $\mathbf{b}_j[j] = 1$ is the only non-zero entry.Let $\mathbf{k} = (k, 2, 2, \ldots, 2)$



We can convert a vector $\mathbf{a} = (\mathbf{a}[0], \ldots, \mathbf{a}[m])$ to a large number. For example, convert $a = (1, 4, 5, 3)$ to number 1453

$$-1453 = \mathbf{a}[0] \times 1000 + \mathbf{a}[1] \times 100 + \mathbf{a}[2] \times 10 + \mathbf{a}[3] \times 1$$

To avoid carry, use N-ary representation instead (for sufficiently large N) Additions with vectors are now equivalent to additions with numbers, since we do not have carry issue.

[SubsetSum+] Given a collection of positive integers $S = \{a_1, \ldots, a_n\}$ and $k \in \mathbb{Z}^+$, decide if there is a sub-collection $T \subseteq S$ such that $\sum_{a_i \in T} a_i = k$.

SubsetSum + is NP-complete

- The same proof for SubsetSum can prove this!

[Partition] Given a collection of integers S, decide if there is a partition of S to A and B such that $\sum_{a \in A} a = \sum_{b \in B} b$.

[Partition+] Given a collection of positive integers S, decide if there is a partition of S to A and B such that $\sum_{a \in A} a = \sum_{b \in B} b$

如何证明 NP-Complete

1. Prove $f \in$ NP.
2. Construct the reduction $g \leq_k f$.
   - Fix an instance x of g. Describe the corresponding f instance y.
3. [Completeness] x is yes $\Rightarrow$ y is yes
4. [Soundness] x is no $\Rightarrow$ y is no. (y is yes $\Rightarrow$ x is yes)

Dominating Set

Given an undirected graph $G = (V, E)$, a dominating set is a subset of vertices S such that, for any $v \in V \backslash S$, there is a vertex $u \in S$ that is adjacent to v.

Theorem 8. *DominatingSet is NP-complete*

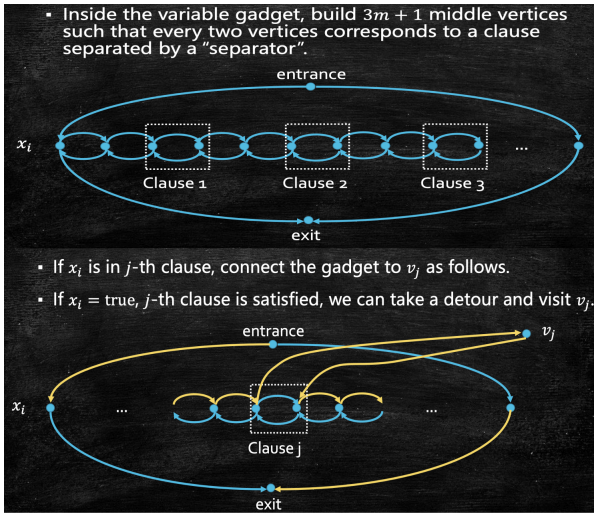
Proof. First of all, DominatingSet is in NP, as a dominating set S can be served as a certificate, and it can be verified in polynomial time whether S is a dominating set and whether $|S| = k$. To show that DominatingSet is NP-complete, we present a reduction from VertexCover. Given a VertexCover instance $(G = (V, E), k)$, we construct a DominatingSet instance $(G' = (V', E'), k')$ as follows. The vertex set is $V' = \bar{V} \cup \bar{E}$, which is defined as follows. For each vertex $v \in V$ in the VertexCover instance, construct a vertex $\bar{v} \in \bar{V} \subseteq V'$; for each edge $e \in E$ in the VertexCover instance, construct a vertex $w_e \in \bar{E} \subseteq V'$. The edge set E' is defined as follows. For each edge $e = (u, v)$ in the VertexCover instance, build two edges $(\bar{u}, w_e), (\bar{v}, w_e) \in E'$. For any two vertices $\bar{u}, \bar{v}$ in $\bar{v}$, build an edge $(\bar{u}, \bar{v})$ Define $k' = k$

Suppose $(G = (V, E), k)$ is a yes VertexCover instance. There exists a vertex cover $S \subseteq V$ with $|S| = k$. We will prove $\bar{S}$ corresponding S is a dominating set in G'. For each vertex in $\bar{V}$, it is covered by any vertex in $\bar{S}$ as $\bar{V}$ forms a clique. For each vertex w_e in $\bar{E}$, let $e = (u, v) \in E$ be the corresponding edge in the VertexCover instance. We have either $u \in S$ or $v \in S$ (or both), as S is a vertex cover. This implies either $\bar{u} \in \bar{S}$ or $\bar{v} \in \bar{S}$ (or both), which further implies w_e is covered as $(\bar{u}, w_e), (\bar{v}, w_e) \in \bar{E}$ by our construction. Since $|\bar{S}| = |S| = k = k'$, the DominatingSet instance we constructed is a yes instance.

Suppose $(G' = (V', E'), k')$ is a yes DominatingSet instance. There exists a dominating set $S' \subseteq V' = \bar{V} \cup \bar{E}$ with $|S'| = k' = k$. We aim to show that $(G = (V, E), k)$ is a yes VertexCover instance. First of all, we can assume $S' \subseteq \bar{V}$ without loss of generality. If we have $w_e \in S'$ for some $w_e \in \bar{E}$, we can replace w_e with either $\bar{u}$ or $\bar{v}$ for the edge $e = (u, v)$ in the VertexCover instance. (In the case $\bar{u}$ and $\bar{v}$ have already been included in S', we can replace w_e with any unpicked vertex in $\bar{V}$.) It is easy to see that S' is still a dominating set after the change, as the set of vertices covered by either $\bar{u}$ or $\bar{v}$ is a superset of the set of vertices covered by w_e (which is just $\{\bar{u}, \bar{v}\}$). Next, since $S' \subseteq \bar{V}, S'$ corresponds to a vertex set $S \subseteq V$ in the VertexCover instance with $|S| = |S'| = k' = k$. It remains to show S is a vertex cover. For any edge $e = (u, v)$, we have either $\bar{u} \in S'$ or $\bar{v} \in S'$ (or both) since S' is a dominating set and $\bar{u}, \bar{v}$ are the only two vertices that can cover w_e. This implies $u \in S$ or $v \in S$ (or both), so S is a vertex cover.

HamiltonianPath

[DirectedHamiltonianPath] Given a directed graph $G = (V, E)$, a source $s \in V$ and a sink $t \in V$, decide if there is a Hamiltonian path from s to t.



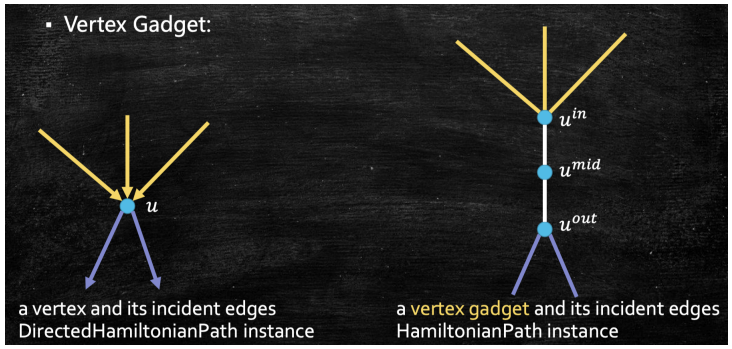
If G' is a yes HamiltonianPath instance, G is a yes Directed HamiltonianPath instance:

Lemma 1. The path in G' must start at s^{in} and end at t^{out}.

Lemma 2. If we first enter a vertex gadget at u^{in} (or u^{out}) we must proceed to u^{mid} and then to u^{out} (or u^{in}).

Lemma 3. The pattern of the path must be in $\rightarrow$ mid $\rightarrow$ out $\rightarrow$ in $\rightarrow$ mid $\rightarrow$ out $\rightarrow \cdots$

- Now we have a Hamiltonian path in G' corresponding to a path in G.



[HamiltonianCycle] Given an undirected graph $G = (V, E)$, decide if G contains a Hamiltonian cycle.HamiltonianCycle is NP-complete.