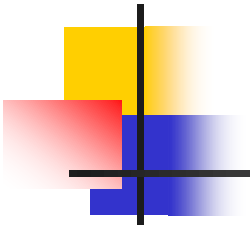
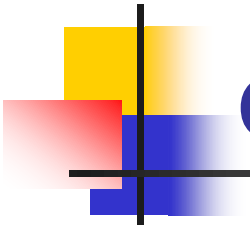


[Chapter 4]
The Greedy Approach

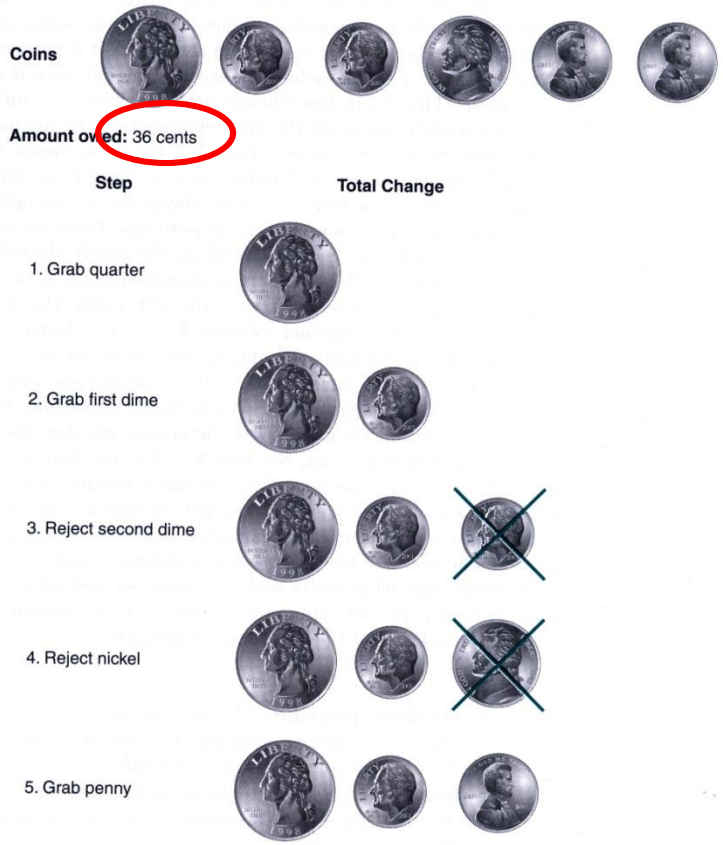


Greedy approach: design process

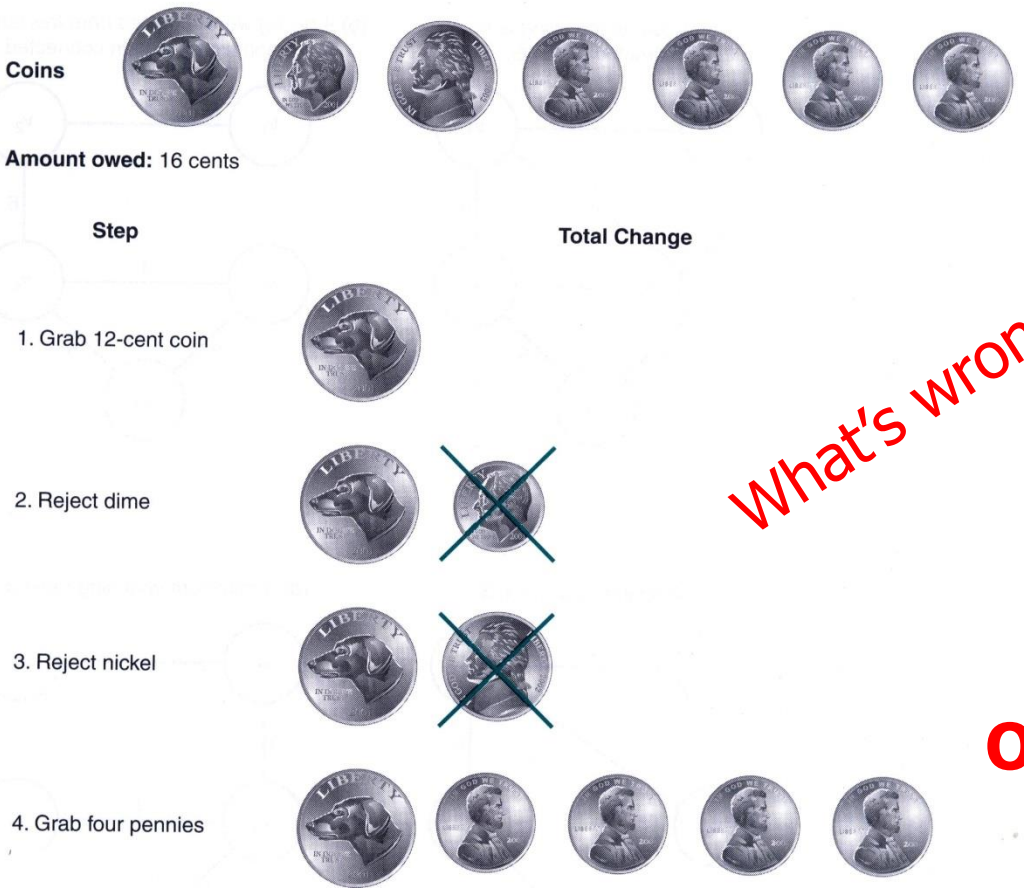
- Dynamic Programming vs. Greedy Approach
  - DP: Recursive property to divide an instance into smaller ones
  - GA: No division into smaller instances, but making a sequence of greedy choices to arrive at a solution.

- Selection procedure
  - Chooses the next item to add to the solution set.
  - Selection is made according to a greedy criterion that satisfies locally optimal consideration at the time.
- Feasibility check
  - Sees if the new set is feasible.
- Solution check
  - Determines if the new set constitutes a solution to the instance.

Example: Coin change problem (a.k.a. Frobenius problem)

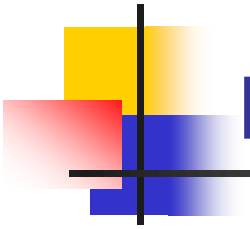


Greedy algorithm is not optimal: check the optimality check when a 12-cent coin is included.



What's wrong??

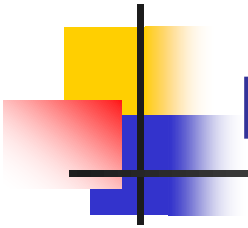
Ooops!



Exercise

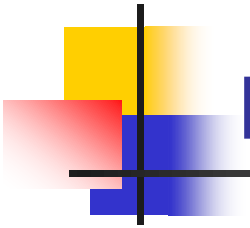
- Greedy approach itself never guarantees the optimal solution. Thus, it is the responsibility of an algorithm designer. *OPTIMALITY PROOF* is a MUST in greedy algorithm design.

- Prove that the greedy algorithm always produces the minimum number of coins for a change if the coin set is {1, 5, 10}.



Minimum spanning tree problem

- Graph theory terms:
  - Undirected graph G: G = (V, E)
  - Connected graph
  - Weighted graph
  - Path
  - Cyclic graph and acyclic graph
  - Subgraph G' = (V', E') if V' (E') is a subset of V (E)
  - Tree is an acyclic and connected graph.
  - Spanning tree for G is an acyclic connected subgraph that contains all the nodes in G.
  - Minimum spanning tree for G is a spanning tree of minimum weight for the graph.

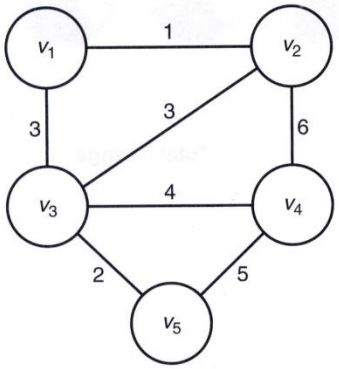


MST Example Applications

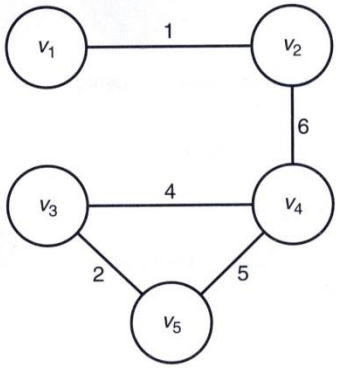
- Interstate highway construction
- Subway, railroad construction
- Telecommunication network construction
- Plumbing, wiring, power transmission, and etc.

Example: A weighted graph and three subgraphs.

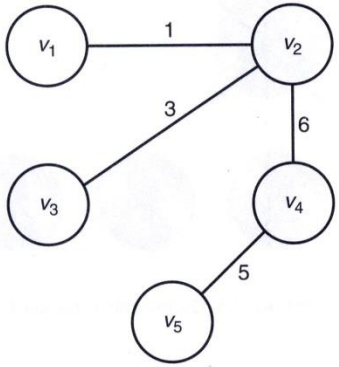
(a) A connected, weighted, undirected graph G.



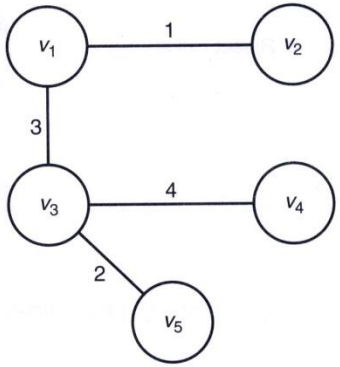
(b) If (v_4, v_5) were removed from this subgraph, the graph would remain connected.



(c) A spanning tree for G.



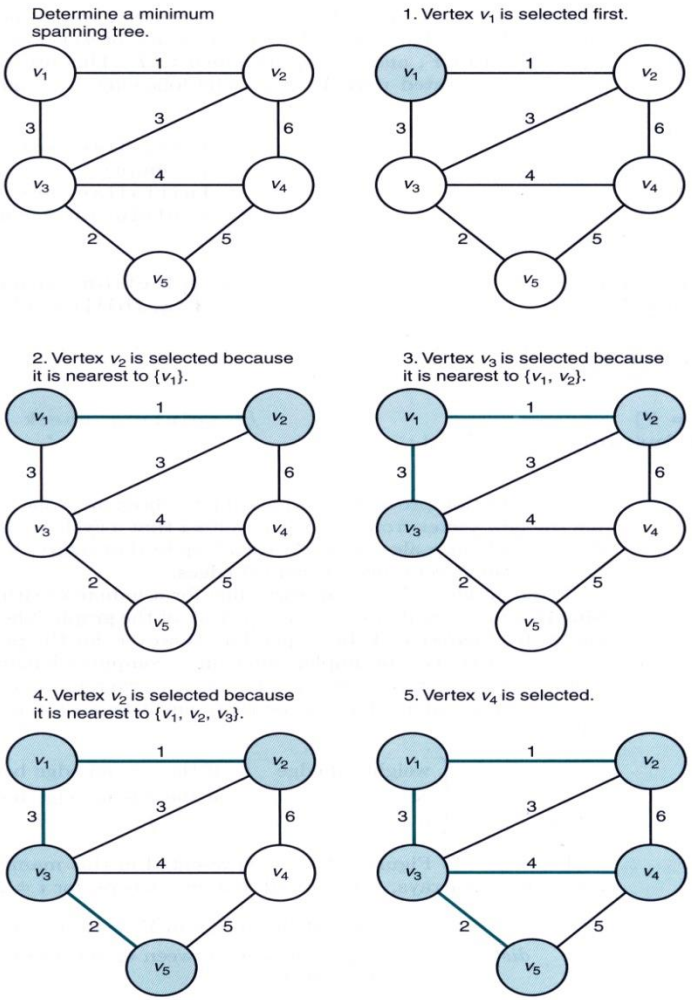
(d) A minimum spanning tree for G.



Prim's algorithm

- Algorithm sketch
  - 1. $F := 0$;
  - 2. $Y := \{v_1\}$;
  - 3. while (*the instance is not solved*) {
    select a vertex in $V - Y$ that is nearest to Y;
    add the vertex to Y;
    add the edge to F;
    if ($Y == V$) *the instance is solved*;
  - }

Step-by-step snapshots in applying Prim's algorithm



Prim's algorithm (Cont'd)

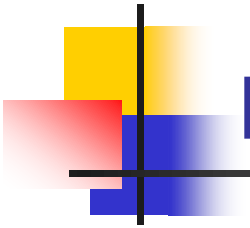
- Adjacency Matrix Representation

$$W[i][j] = \begin{cases} weight & \text{If there is an edge from } v_i \text{ to } v_j \\ \infty & \text{If there is no edge from } v_i \text{ to } v_j \\ 0 & \text{If } i = j. \end{cases}$$

- Two additional arrays
  - $nearest[i]$: index of the vertex in Y nearest to v_i.
  - $distance[i]$: weight on edge between v_i and the vertex indexed by $nearest[i]$.

The array representation *W* of the graph in finding MST

	1	2	3	4	5
1	0	1	3	∞	∞
2	1	0	3	6	∞
3	3	3	0	4	2
4	∞	6	4	0	5
5	∞	∞	2	5	0



Prim's algorithm analysis

- Time complexity analysis
  - Basic operation
    - There are two loops, each with n-1 iterations, inside the loop. Executing the instructions inside each of them can be considered to be doing the basic operation once.
  - Input size
    - n, the number of vertices.
  - $T(n) = 2(n\text{-}1)(n\text{-}1) \in \Theta(n^2)$

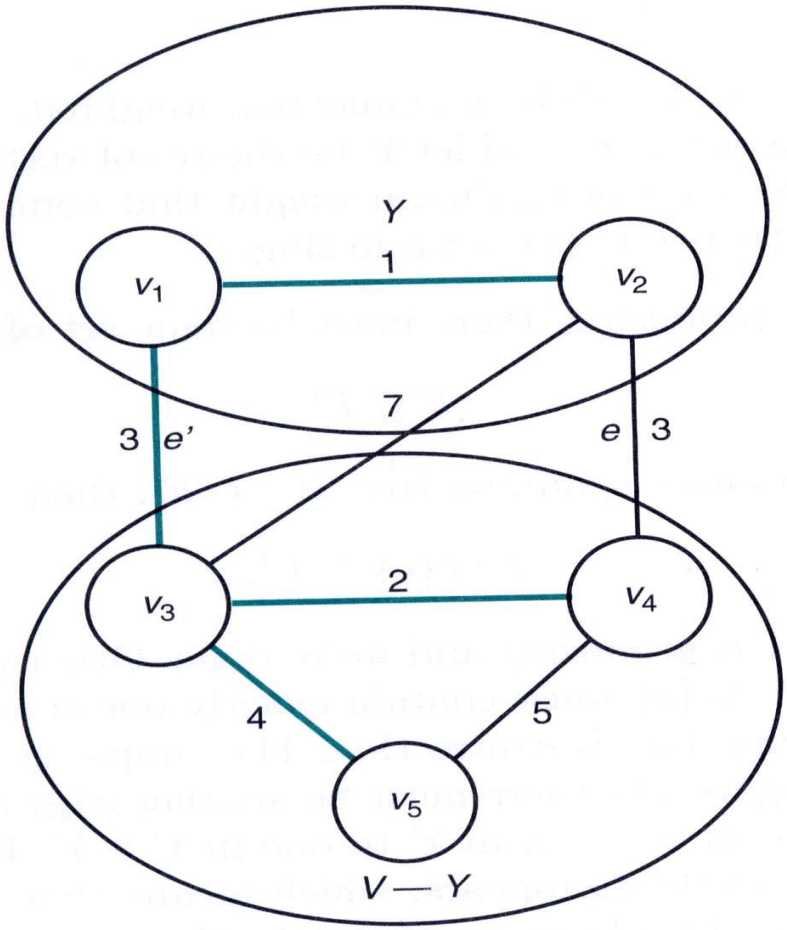
Optimality Proof for Prim's algorithm

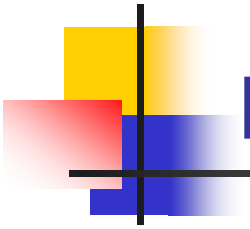
- In $G = (V, E)$, a subset F of E is called *promising* if edges can be added to it so as to form a MST.
- Lemma 4.1 Let G be a connected, weighted, undirected graph; let F be a promising subset of E; and let Y be the set of vertices connected by the edges in F. If e is an edge of minimum weight that connects a vertex in Y to a vertex in $V - Y$, the $F \cup \{e\}$ is promising.
- Theorem 4.1 Prim's algorithm always produces a MST.
  - Induction base: Clearly the empty set is promising.
  - I. H. Assume that, after a given iteration of the loop, the set of edges so far selected, namely F, is promising.
  - I. S. Let's show that the $F \cup \{e\}$ is promising. Because the edge e selected in the next iteration is an edge of minimum weight that connects Y to $V - Y$, $F \cup \{e\}$ is promising, by Lemma 4.1

Proof for Lemma 4.1

- Because F is *promising*, there must be some set of edges F' such that $F \subseteq F'$ and (V,F') is a MST.
- Case 1: If $e \in F'$, $F \cup \{e\} \subseteq F'$, therefore $F \cup \{e\}$ is promising.
- Case 2: If $e \notin F'$, $F' \cup \{e\}$ should contain at least one cycle, and e is an edge in the cycle. Then, there must be another edge $e' \in F'$ that connects Y to $V - Y$. If we remove e' from $F' \cup \{e\}$, the cycle will disappear, which means that we have a spanning tree. Because e is an edge of minimum weight that connects Y to $V - Y$, the weight of e must be less than or equal to the weight of e' (in fact, they must be equal). So, $F' \cup \{e\} - \{e'\}$ is an MST. Now $F \cup \{e\} \subseteq F' \cup \{e\} - \{e'\}$, because e' cannot be in F (recall that edges in F connect only vertices in Y.) Therefore, $F \cup \{e\}$ is promising, which completes the proof of Lemma 4.1.

Figure 4.6 A graph illustrating the proof in Lemma 4.1. The edges in F′ are shaded in color





Kruskal's algorithm

- Algorithm sketch
  - 1. $F := 0$; // the set of edges
  - 2. Create disjoint subsets of V, one for each vertex and containing only that vertex;
  - 3. Sort the edges in E in nondecreasing order;
  - 4. while (*the instance is not solved*) {

        select next edge;

        if (the edge connects two vertices in disjoint subsets) {

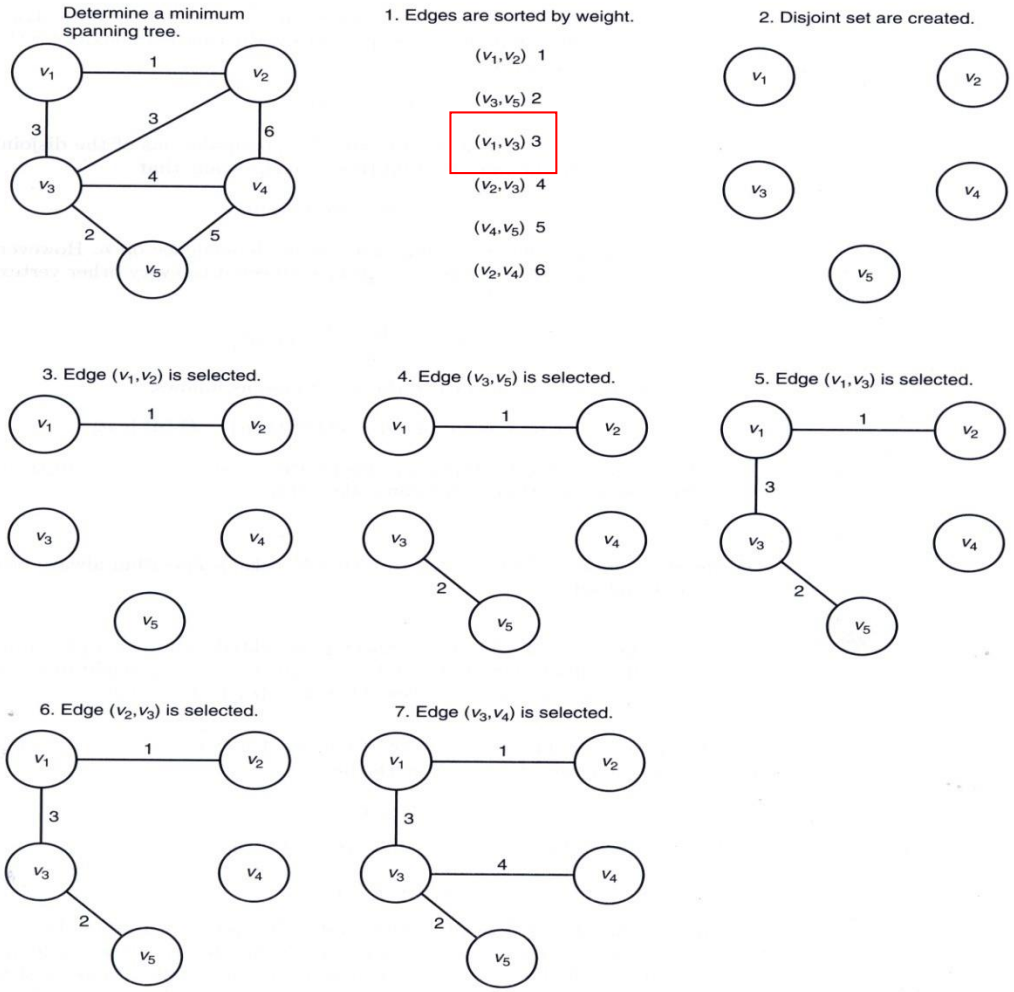
          merge the subsets;

          add the edge to P;

        }

        if (all the subsets are merged) *the instance is solved*;
  - }

A weighted graph (in upper-left corner) and the Kruskal's algorithm for that graph.



Kruskal's algorithm analysis

- Time complexity analysis
  - Basic operation: a comparison operation.
  - Input size: n, m, the number of vertices and edges, respectively.
  - The time to sort the edges. (Recall the mergesort algorithm) $\Theta(m \lg m)$
  - The time in the loop: $\Theta(m \lg m)$
  - The time to initialize n disjoint sets: $\Theta(n)$
  - Since $m \geq n - 1$, $W(m, n) = \Theta(m \lg m)$
  - But in worst case, $m = \frac{n(n-1)}{2} \in \Theta(n^2)$

$$W(m,n) \in \Theta(n^2 \lg n^2) = \Theta(2n^2 \lg n) = \Theta(n^2 \lg n)$$

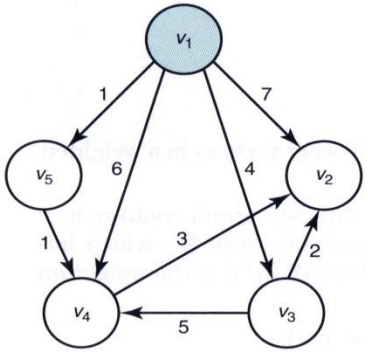
- Optimality proof is very similar to Prim's case.

Comparing Prim's algorithm with Kruskal's algorithm

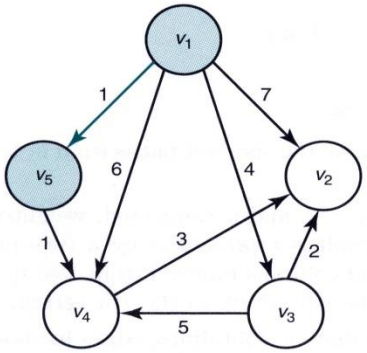
	$W(m,n)$	sparse graph	dense graph
Prim	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Kruskal	$\Theta(m \lg m)$ and $\Theta(n^2 \lg n)$	$\Theta(n \lg n)$	$\Theta(n^2 \lg n)$

Dijkstra's algorithm (shortest paths)

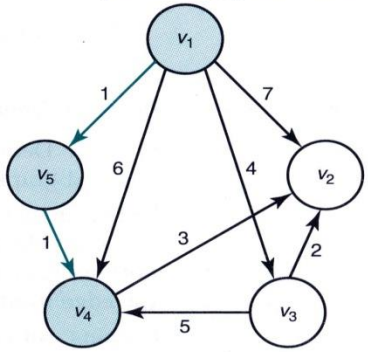
Compute shortest paths from v_1.



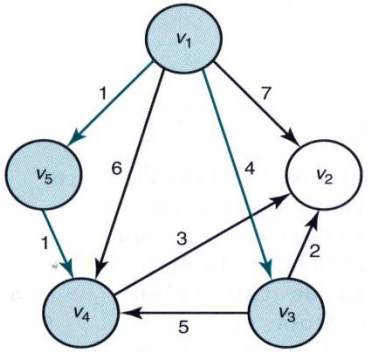
1. Vertex v_5 is selected because it is nearest to v_1.



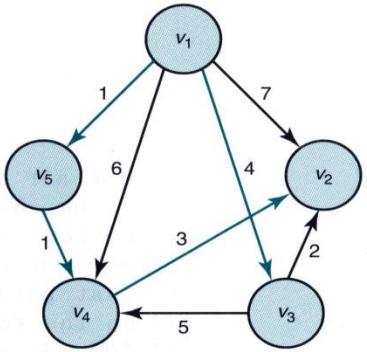
2. Vertex v_4 is selected because it has the shortest path from v_1 using only vertices in $\{v_5\}$ as intermediates.

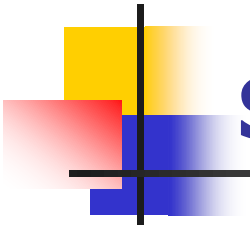


3. Vertex v_3 is selected because it has the shortest path from v_1 using only vertices in $\{v_4, v_5\}$ as intermediates.



4. The shortest path from v_1 to v_2 is $[v_1, v_5, v_4, v_2]$.



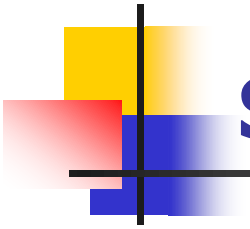


Scheduling

- Minimizing total time in the system

Job	*Time in the system*

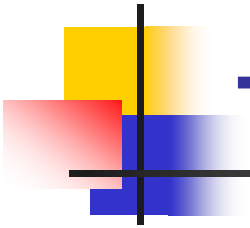
- Time complexity:



Scheduling

- Scheduling with deadlines

- Solution:



The Knapsack Problem

- Problem
- Goal
- Brute force algorithm complexity
- Greedy algorithm