Analysis and Design of Algorithms

Chapter 9: Greedy Algorithms

School of Software Engineering © Ye Luo

Content

- ***The Introduction of Greedy Algorithms***
  - *Change Making Problem*
  - *Definition of Greedy Algorithm*
  - *Category of Greedy Algorithms*
- ***Problems solved by Greedy Algorithms***
  - *Fractional Knapsack Problem*
  - *Single-source Shortest Paths problem*
  - *Minimum Spanning Tree*

Change Making Problem

- ***Problem Description (找零问题)***

*Given unlimited amounts of coins $d_1 > \ldots > d_m$, give the change for **amount X** with the **least number of coins**.*

- ***Idea*** *---"Greedy" thinking*
  - Take the coin with largest denomination without exceeding the remaining amount of cents, it reduces the remaining amount the most
  - Make the locally best choice at each step

Change Making Problem

- Example 1

How to make 48 cents of change using coins of

d1=25 , d2=10 , d3=5 , d4=1

so that the total number of coins is the smallest?

Solution: 25, 10, 10, 1, 1, 1

What is the greedy strategy?

Definition of Greedy Algorithms

- **Greedy Strategy**
  - *A greedy algorithm makes a **locally optimal choice** step by step **in the hope** that a sequence of such choice will **lead to a globally optimal solution**.*
    - *Constructing a solution through a sequence of steps*
    - *Expanding a partially constructed solution obtained so far at this step*
    - *Until a complete solution to the problem is reached*
  - *The choice made at each step must be:*

Feasible	*-Satisfy the problem's constraints*
Locally optimal	*-Be the best local choice among all feasible choices*
Irrevocable	*-Once made, choice can't be changed on subsequent steps.*

Basic Steps of Greedy Algorithms

- **General features of the problems solved by Greedy Alg.**
  - ✓ The solution to the problem consists of a subset of **N** sequential inputs that satisfy some constraints. (问题的解由n个满足条件的序列输入构成)

- **Basic steps of greedy method**

找零问题中的钱币面值

找钱超过应找数量

① According to the question, choose a metric and sort the N inputs by this metric.

② If x_n (n<=N, the nth input) and the partial optimal solution($\{x_1, x_2, \ldots, x_{n-2}\}$) cannot be a feasible solution under this metric, then x_n is deleted and move to x_{n+1}.

③ Otherwise, merge the current input into the partial solution to get a new partial solution ($\{x_1, x_2, \ldots, x_{n-2}, x_n\}$).

④ This process continues until all N inputs have been considered, and then the subset of inputs recorded in the optimal solution set constitutes the optimal solution to the problem under this metric.

Change Making Problem

- **Example 1**

How to make 48 cents of change using coins of

d1=25 , d2=10 , d3=5 , d4=1

so that the total number of coins is the smallest?

Solution: 25, 10, 10, 1, 1, 1

- **Is the solution globally optimal?**

Yes

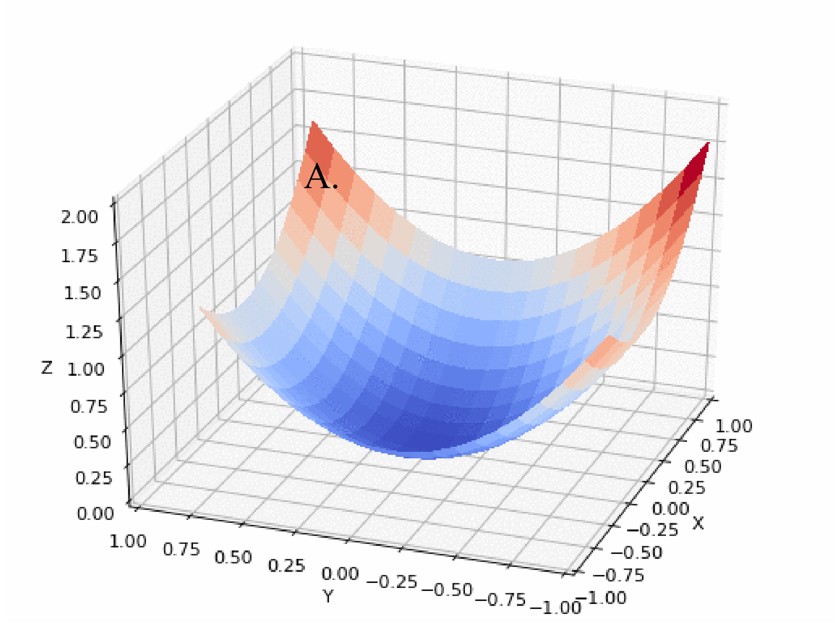
- **Change Making Example 2**

How to make ***41 cents*** *of change using coins of*
d1=25 , d2=20 , d3=10 , d4=5, d5= 1
so that the total number of coins is the smallest?

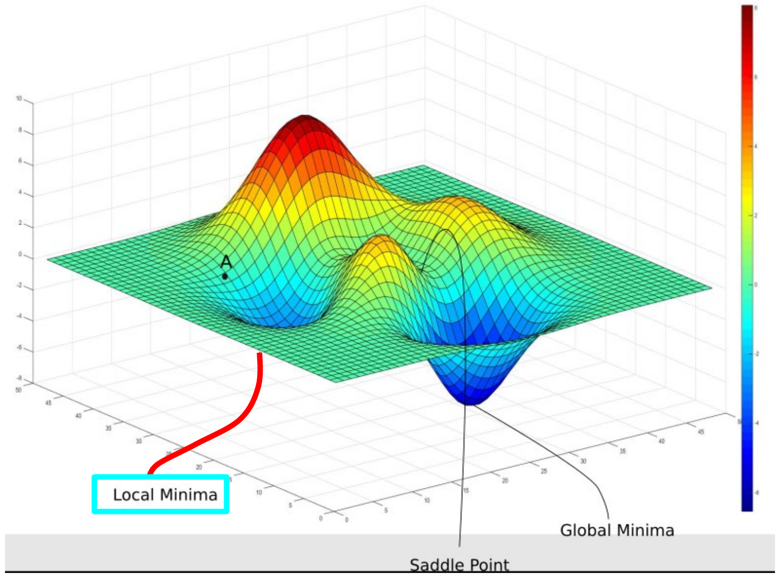
***Greedy Solution**: 25, 10, 5, 1*

No, *optimal solution: 20, 20, 1*

Greedy Algorithm v.s. Wisdom of Ancient Chinese

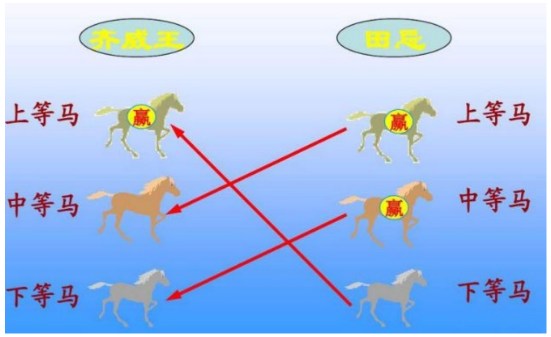


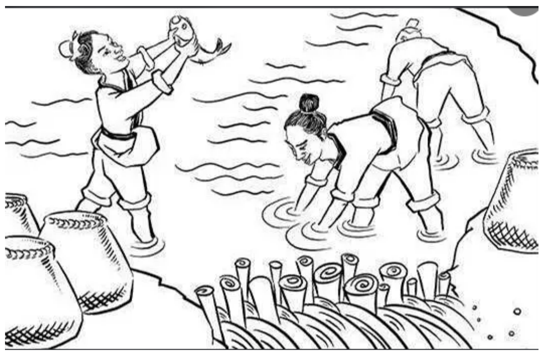
v.s.



Ideal case

Practical case





做人做事要顾全大局，不能只贪图眼前的一点利益，要有大局意识。

Category of Greedy Algorithms

- ***For some problems, yields an optimal solution for every instance.***
  - Some instances of change making
  - Minimum Spanning Tree (MST)
  - Single-source shortest paths
  - Huffman codes

- ***For most, does not, but can be useful for fast approximations.***
  - Traveling Salesman Problem (TSP)
  - Knapsack problem
  - Other optimization problems

贪心策略解背包问题

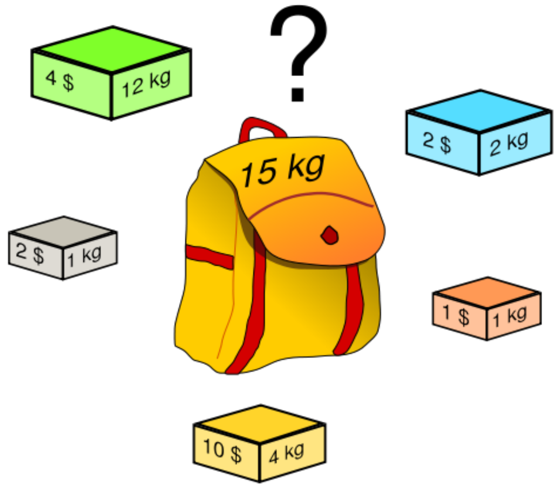
- 背包问题：与0-1背包问题类似，给定n 种物品和一背包。物品 i 的重量是w_i，其价值为v_i，背包的容量为C。**假定将物品 i 的某一部分x_i放入背包就会得到v_ix_i的效益($0\le x_i \le 1$, $v_i>0$)**。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?
    - **所不同的是在选择物品i装入背包时，可以选择物品i的一部分，而不一定要全部装入背包，$1\le i\le n$。**
    - 其形式化描述是：

      给定$c>0, w_i>0, v_i>0, 1\le i\le n$,

      要找出一个n元向量$(x_1,x_2,\ldots,x_n)$,

      使得 $$\max\sum_{i=1}^{n} v_i x_i$$

$$\sum_{i=1}^{n} w_i x_i \le c \qquad 0\le x_i\le 1,\ 1\le i\le n$$



贪心策略解分数背包问题

- ***Example:***

设，n=3，C=20，$(v_1,v_2,v_3) = (25,24,15)$， $(w_1,w_2,w_3) = (18,15,10)$。

可能的可行解有：

	(x_1,x_2,x_3)	$\sum w_i x_i$	$\sum v_i x_i$	
①	(1/2,1/3,1/4)	16.5	24.25	//没有装满背包//
②	(1, 2/15, 0)	20	28.2	
③	(0, 2/3, 1)	20	31	
④	(0, 1, 1/2)	20	31.5	

- 贪心策略下，以什么为度量标准，对物品进行排序依次放入背包呢，即贪心选择呢？

分数背包问题

度量标准一：每装入一件物品，使背包获得最大的价值增量。

- 该度量标准下的处理规则是：
  - 按物品**价值的非增**次序将物品一件件地放入到背包；
  - 如果正在考虑的物品放不进去，则只取其一部分装满背包：

贪心策略解分数背包问题

- ***Example:***
  - n=3，C=20，$(v_1,v_2,v_3) = (25,24,15)$， $(w_1,w_2,w_3) = (18,15,10)$。

    ∵ $v_1>v_2> v_3$

    - ∴ 首先将物品1放入背包，此时$x_1=1$，背包获得$v_1=25$的利益增量，同时背包容量减少$w_1=18$个单位，剩余空间$\Delta C=2$。
    - 其次考虑物品2和3。就$\Delta C=2$而言有，只能选择物品2或3的一部分装入背包。
      - ✓ 物品2： 若 $x_2=2/15$， 则 $v_2\,x_2=16/5=3.1$
      - ✓ 物品3： 若 $x_3=2/10$， 则 $v_3\,x_3=3$
      - ✓ 为使背包的效益有最大的增量，应选择物品2的2/15装包，即$x_2=2/15$
    - 最后，背包装满， $\Delta C=0$，物品3不装包，即$x_3=0$ 。
    - 背包最终可以获得利益值$= x_1\,v_1 + x_2\,v_2 + x_3\,v_3 = 28.2$ (**次优解,**非问题的最优解)

贪心策略解分数背包问题

- **度量标准二：以容量作为度量标准**

  Problem: 以目标函数作为度量标准所存在的问题：尽管背包的利益值每次得到了最大的增加，但背包容量也过快地被消耗掉了，从而不能装入“更多”的物品。

  Solution: 让背包容量尽可能慢地消耗，从而可以尽量装入“较多”的物品。

  即，以容量作为度量

  - 该度量标准下的处理规则：
    - 按物品重量的非降次序将物品装入到背包；
    - 如果正在考虑的物品放不进去，则只取其一部分装满背包；

贪心策略解分数背包问题

- ***Example:***
  - $n=3$，$C=20$，$(v_1,v_2,v_3) = (25,24,15)$， $(w_1,w_2,w_3) = (18,15,10)$。

    ∵ $w_3<w_2< w_1$

    - ∴ 首先将物品3放入背包，此时$x_3=1$，背包获得$v_3=15$的利益增量，同时背包容量减少$w_3=10$个单位，剩余空间$\Delta C=10$。
    - 其次考虑物品2和1。就$\Delta C=10$而言有，也只能选择物品2或1的一部分装入背包。
      - ✓ 物品2： 若 $x_2=10/15$， 则 $v_2 x_2=16$
      - ✓ 物品1： 若 $x_1=10/18$， 则 $v_1 x_1=13.9$
      - ✓ 为使背包的效益有最大的增量，应选择物品2的10/15装包，即$x_2=10/15$
    - 最后，背包装满$\Delta C=0$，物品1将不能装入背包，故 $x_1=0$ 。
    - 背包最终可以获得利益值$= x_1 v_1 +x_2 v_2+x_3 v_3 = 31$ (次优解,非问题的最优解)

存在的问题：效益值没有得到“最大程度”的增加

贪心策略解分数背包问题

- **度量标准三：单位重量价值增量**
  - 影响背包利益值的因素：
    - ☞ 背包的容量*C*
    - ☞ 放入背包中的物品的重量及其可能带来的利益值
  - 可能的策略是：在背包利益值的增长速率和背包容量消耗速率之间取得平衡，即每次装入的物品应使它所占用的每一单位容量能获得当前最大的单位利益。
  - ⇨ 这种策略下的量度：已装入的物品的累计利益值与所用容量之比。
  - ***Solution:*** 新的量度标准是：每次装入要使累计利益值与所用容量的比值有最多的增加（首次装入）和最小的减小（其后的装入）。

    此时，将按照物品的**单位利益值：v_i/w_i 比值的非增次序**考虑。

贪心策略解分数背包问题

- ***Example:***
  - n=3，C=20，$(v_1,v_2,v_3) = (25,24,15)$， $(w_1,w_2,w_3) = (18,15,10)$。
    - ∵ $v1/w1 < v3/w3 < v2/w2$ ∴ 首先将物品2放入背包，此时$x_2=1$，背包获得$v_2=24$的利益增量，同时背包容量减少$w_2=15$个单位，剩余空间$\Delta C=5$。
    - 其次，在剩下的物品1和3中，应选择物品3,且就$\Delta C=5$而言有，只能放入物品3的一部分到背包中 。即 $x_3=5/10=1/2$;
    - 最后，背包装满$\Delta C=0$，物品1将不能装入背包，故$x_1=0$ 。
    - 最终可以获得的背包利益值＝$x_1 v_1 + x_2 v_2 + x_3 v_3 = 31.5$ (最优解)

算法可视化 Demo

贪心策略解分数背包问题

- GreedyKnapsack算法描述
  - 用贪心算法解连续背包问题的基本步骤：
    - 计算每种物品单位重量的价值v_i/w_i，
    - 依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包。
    - 若将这种物品全部装入背包后，背包内的物品总重量未超过C，则选择单位重量价值次高的物品并尽可能多地装入背包。
    - 依此策略一直地进行下去，直到背包装满为止。

贪心策略解分数背包问题

贪心算法解连续背包问题的效率分析

- 算法GreedyKnapsack的主要计算时间在于将各种物品依其单位重量的价值**从大到小排序，**排序算法的时间上界为***O(nlogn)***。
- 依次放入物品的时间复杂度是***O(n)***。
- 因此，GreedyKnapsack算法的计算时间上界为***O(nlogn)***。

贪心解 的正确性证明

- **Greedy算法的正确性证明**

  证明分数背包问题由第三种策略所得到的**贪心解**是问题的**最优解。**

- **怎么证明?**

  首先，考察问题的一个整体最优解，并证明可修改这个最优解，使其以贪心选择（**贪心选择性**）开始。

  而且，作了贪心选择后，原问题简化为一个规模更小的类似子问题。

  然后，用数学归纳法证明，通过每一步作贪心选择，最终可得到问题的一个整体最优解。

  其中，证明贪心选择后的问题简化为规模更小的类似子问题的关键在于利用该问题的**最优子结构性质**。

贪心解 是最优解的证明

- 贪心选择性质和最优子结构

① **贪心选择性质（Greedy-choice property）：**若原问题的整体最优解可以通过一系列局部最优的选择得到，即贪心选择来达到，则该问题称为具有贪心选择性质。

② **最优子结构性质（ Optimal substructure ）：**若一个问题的最优解包括它的子问题的最优解，则称其具有最优子结构性质。

贪心算法解分数背包问题

- **定理**

如果$p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$, 则算法GreedyKnapsack对于给定的背包问题实例生成一个最优解。

- **证明：**

  - 设$X=(x_1, x_2, \ldots, x_n)$是GreedyKnapsack所生成的贪心解。
  - 如果所有的x_i都等于1，则显然X就是问题的最优解。
  - 否则， 设j是使$x_j \neq 1$的最小下标。由算法的执行过程可知，

    $x_i = 1$, $1 \leq i < j$,

    $x_j = [0, 1]$

    $x_i = 0$, $j < i \leq n$

贪心算法解分数背包问题

- 设$Y = (y_1, y_2, \ldots, y_n)$是问题的最优解，且有 $\sum w_i y_i = C$。
- 若$X = Y$，则X就是最优解。
- 否则，X和Y至少在1个分量上存在不同。设k是使得$y_k \neq x_k$的最小下标，则有$y_k < x_k$。可分以下情况说明：
  - a)若$k < j$，则$x_k=1$。因为$y_k \neq x_k$，从而有$y_k < x_k$
  - b) 若$k = j$，由于 $\sum w_i x_i = C$ ，有 $1 \le i < j = k$ 或者 $k = j < i \le n$，*则有*

I. $k = j < i \le n$

$\Rightarrow x_j \in [0,1],\ x_j = x_k \in [0,1] \,\&\, x_i = 0$

$if\ \ x_k < y_k \,\&\, \sum x_i w_i = C$

then

$y_k \in [0,1] \,\&\, x_k < y_k \Rightarrow y_{k+1} = 0\ \ \&\sum y_i w_i > C$

$or\ \ y_k = 1, y_{k+1} \in [0,1] \Rightarrow y_{k+2} = 0\ \ \&\sum y_i w_i > C$

$or\ \ y_k = 1, y_{k+1} = 1, y_{k+2} \in [0,1] \Rightarrow y_{k+3} = 0\ \ \&\sum y_i w_i > C$

…

Contradiction

贪心算法解分数背包问题

- 设$Y = (y_1, y_2, \ldots, y_n)$是问题的最优解，且有 $\sum w_i y_i = C$。
- 若$X = Y$，则X就是最优解。
- 否则，X和Y至少在1个分量上存在不同。设k是使得$y_k \neq x_k$的最小下标，则有$y_k < x_k$。可分以下情况说明：

  a)若$k < j$，则x_k=1。因为$y_k \neq x_k$，从而有$y_k < x_k$

  b) 若$k = j$，由于 $\sum w_i x_i = C$ ，有 $1 \leq i < j = k$，或者 $k = j < i \leq n$ *则有*

$$
\begin{aligned}
&\text{II.}\quad k = j < i \leq n \\
&\Rightarrow x_j \in [0,1],\ x_j = x_k \in [0,1] \,\&\, x_i = 0 \\
&\mathit{if}\ x_k < y_k \,\&\, \sum x_i w_i = C \\
&\mathit{then} \\
&y_k \in [0,1] \,\&\, x_k < y_k \Rightarrow y_{k+1} = 0 \ \ \&\sum y_i w_i > C \\
&\mathit{or}\ \ y_k = 1, y_{k+1} \in [0,1] \Rightarrow y_{k+2} = 0 \ \ \&\sum y_i w_i > C \\
&\mathit{or}\ \ y_k = 1, y_{k+1} = 1, y_{k+2} \in [0,1] \Rightarrow y_{k+3} = 0 \ \ \&\sum y_i w_i > C \\
&\ldots \\
&\text{Contradiction}
\end{aligned}
$$

  c)若$k > j$，则 $\sum w_i y_i > C$ ，不能成立。

贪心算法解分数背包问题

- 设$Y = (y_1, y_2, \ldots, y_n)$是问题的最优解，且有 $\sum w_i y_i = C$。
- 若$X = Y$，则X就是最优解。
- 否则，X和Y至少在1个分量上存在不同。设k是使得$y_k \neq x_k$的最小下标，则有$y_k < x_k$。可分以下情况说明：

  a)若$k < j$，则x_k=1。因为$y_k \neq x_k$，从而有$y_k < x_k$

  b) 若$k = j$，由于 $\sum w_i x_i = C$ ，且 对$1 \le i < j = k$，有$y_i = x_i = 1$，而对$k=j < i \le n$, 有 $x_i = 0$；故此时若$y_k > x_k$，则将有 $\sum w_i y_i > C$ ，与Y是可行解相矛盾。而$y_k \neq x_k$, 所以$y_k < x_k$。

  c)若$k > j$，则 $\sum w_i y_i > C$ ，不能成立。

- Y中作以下调整：将y_k增加到x_k，为保持解的可行性，必须从$(y_{k+1}, \ldots, y_n)$中减去同样多的量。设调整后的解为$Z=(z_1, z_2, \ldots, z_n)$，其中$z_i = x_i$，$1 \le i \le k$，且有：$\sum_{k<i\le n} w_i (y_i - z_i) = w_k (z_k - y_k)$

贪心算法解分数背包问题

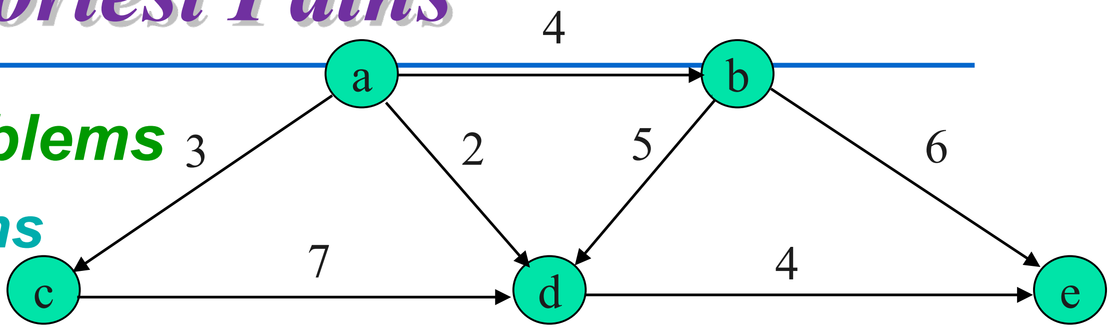
Z的利益值有：
$$\sum_{1\le i\le n} v_i z_i = \sum_{1\le i\le n} v_i y_i + (z_k - y_k) w_k v_k / w_k - \sum_{k<i\le n} (y_i - z_i) w_i v_i / w_i$$
$$\ge \sum_{1\le i\le n} v_i y_i + [(z_k - y_k) w_k - \sum_{k<i\le n} (y_i - z_i) w_i] v_k / w_k$$
$$= \sum_{1\le i\le n} v_i y_i$$

由以上分析得，

- 若 $\sum v_i z_i > \sum v_i y_i$ ，则Y将不是最优解；
- 若 $\sum v_i z_i = \sum v_i y_i$ ，且Z=X，则X就是最优解；

或者Z≠X，则重复以上替代过程，或者证明Y不是最优解，或者把Y转换成 X，从而证明X是最优解。

Single-source Shortest Paths

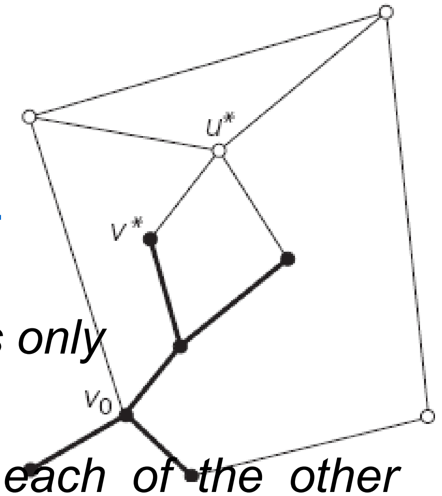


Shortest Path Problems

- *All pair shortest paths*
  - *Floy's algorithm*
- *Single Source Shortest Paths*
  - *Dijkstra's algorithm*
  - *Given a weighted graph G, find the shortest paths from a source vertex s to each of the other vertices*
    - *not a single shortest path that starts at the source and visits all the other vertices. e.g. traveling salesman*
- *Applications*

  - transportation planning
  - packet routing in communication networks
  - finding shortest paths in social networks
  - speech recognition, document formatting, robotics, compilers
  - airline crew scheduling.
  - path-finding in video games

Single-source Shortest Paths



Dijkstra's alg.

- *undirected and directed graphs with nonnegative weights only*
- *one source vertex s*
- *to find the shortest paths from a source vertex s to each of the other vertices*

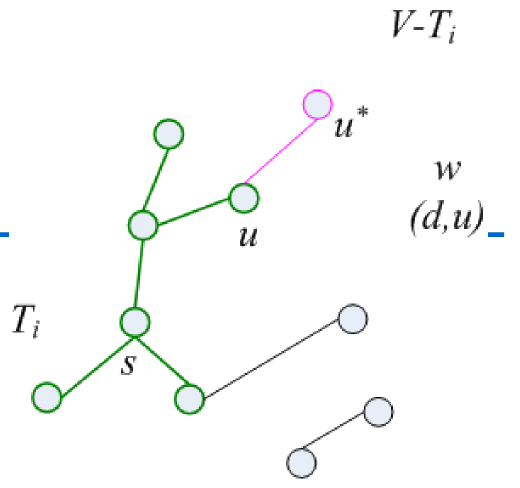
Idea of Dijkstra's

- *Start with a subtree consisting of a single source vertex*
- *a sequence of expanding subtrees, T_1, T_2, … --- one vertex/edge at a time*
  - *In general, before its ith iteration, the alg. has already identified the shortest paths to i -1 other vertices nearest to the source. These vertices, the source, and the edges of the shortest paths leading to them from the source form a subtree T_{i-1} of the given graph.*
  - *Keep track of shortest path from source to each of the vertices in T_i*

Single-source Shortest Paths

- **Dijkstra's alg.**
  - Idea of Dijkstra's
    - On each iteration, $T_i \rightarrow T_{i+1}$ connecting a vertex in tree (T_i) to one not yet in tree
      - expands the current tree in the greedy
    - the next vertex nearest to the source can be found among the vertices adjacent to the vertices of T_i .
      - referred to as "**fringe vertices**"; candidates from which Dijkstra's algorithm selects the next vertex nearest to the source.
      - Actually, all the other vertices can be treated as fringe vertices connected to tree vertices by edges of infinitely large weights

Single-source Shortest Paths



Dijkstra's alg.

Idea of Dijkstra's ('cont)

- *label each vertex w with two labels.*
    - *The numeric label d indicates the length of the shortest path from the source to this vertex found by the algorithm so far*
    - *The other label u indicates the name of the next-to-last vertex on such a path, i.e., the parent of the vertex in the tree being constructed.*
- *finding the next nearest vertex u∗ becomes a simple task of finding a fringe vertex with the smallest d value.*

$$u^* = \min_{w \in V - T_i} d[w] = \min_{w \in V - T_i} (d[u] + c(w,u))$$

  the sum of
    - *the distance to the nearest tree vertex u , i.e. weight of the edge* ***c(w, u)***
    - *and, the length of the shortest path from the source to u (previously determined by the algorithm), i.e.* ***d[u]***

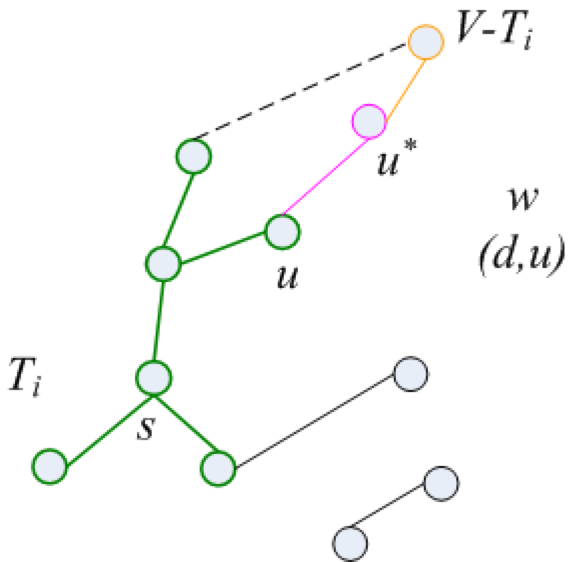
  → *edge (u*,u) with lowest d[u] + c(u*, u)*
- *stops when all vertices are included.*

Single-source Shortest Paths

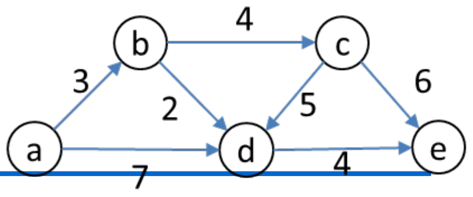
Dijkstra's alg.

Idea of Dijkstra's ('cont)

- *After we have identified a vertex u∗ to be added to the tree, make some modification to other fringe vertices*
  - *Move u∗ from the fringe to the set of tree vertices.*
  - *For each remaining fringe vertex w that is connected to u∗ by an edge of weight c(u∗, w) such that d[u∗] + c(u∗, w) < d[w], update the labels of w by u∗ and d[u∗] + c(u∗, w), respectively.*

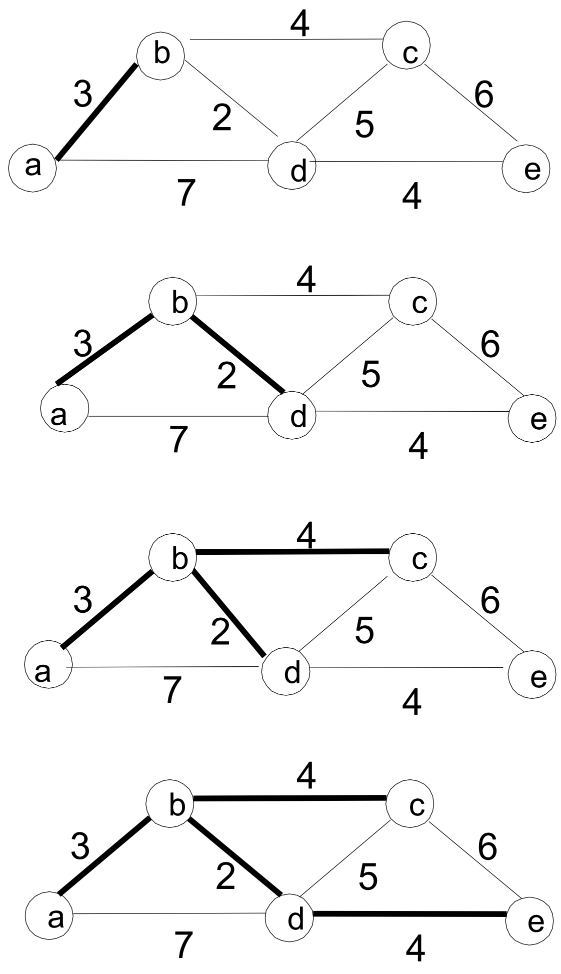


Single-source Shortest Paths



- *Example : Dijkstra's*

Tree vertices	Remaining vertices
a(-,0)	b(a,3) c(-,∞) d(a,7) e(-,∞)
b(a,3)	c(b,3+4) d(b,3+2) e(-,∞)
d(b,5)	c(b,7) e(d,5+4)
c(b,7)	e(d,9)
e(d,9)	



The shortest paths (identified by following nonnumeric labels backward from a destination vertex in the left column to the source) and their lengths (given by numeric labels of the tree vertices) are as follows:

from a to b : a − b of length 3

from a to d : a − b − d of length 5

from a to c : a − b − c of length 7

from a to e : a − b − d − e of length 9

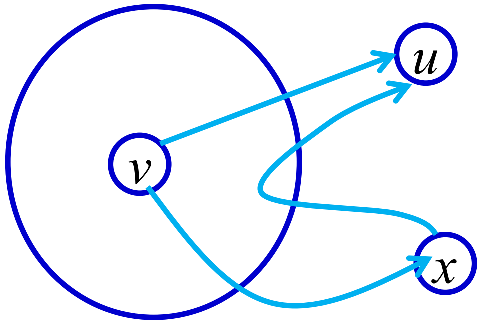
Single-source Shortest Paths

Dijkstra 算法的贪心选择性质

- Dijkstra 算法的贪心选择是从*V-S*中选择具有最短特殊路径的顶点*u*，从而确定从源v_0到*u*的最短路径长度dist[*u*] 。
- 是最优解吗？有无从源v_0到*u*的更短的其它路径？
    - 如果存在一条从源v_0到*u*且长度比dist[*u*]更短的路，设这条路初次走出*S*外到达的顶点为$x \in V\text{-}S$，然后徘徊于*S*内外若干次，最后离开*S*到达*u*。

      $\text{dist}[x] \le d(v,x)$

      $d(v,x) + d(x,u) = d(v,u) < \text{dist}(u)$

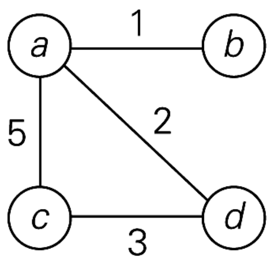


    - 利用边的非负性可知$d(x,u) \ge 0$， ⇨ $\text{dist}[x] < \text{dist}[u]$。
    - 矛盾！

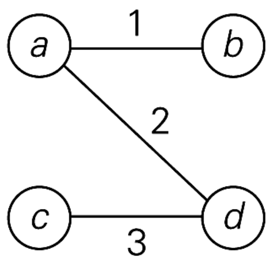
Minimum Spanning Tree

- ***Problem:***
  - *G =(V,E): undirected connected graph。*

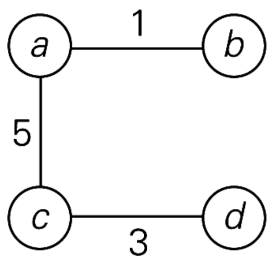
    edge weights for (v,w)∈ E
  - *Spanning Tree: a connected acyclic subgraph (tree) of G that includes all of G's vertices*
    - *Note: a spanning tree with n vertices has exactly n-1 edges.*
  - *weight of a tree: the sum of the weights on all its edges*
  - *minimum spanning tree: spanning tree of the smallest weight*



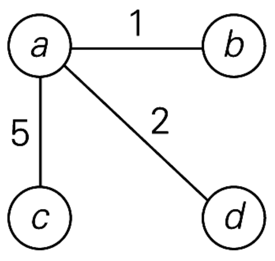
graph



w(T_1) = 6



w(T_2) = 9



w(T_3) = 8

Minimum Spanning Tree

Applications

- problem: given n points, connect them in the cheapest possible way so that there will be a path between every pair of points.
- design of all kinds of networks by providing the cheapest way to achieve connectivity
  - *communication, computer, transportation, electrical*
- identifies clusters of points in data sets
- classification purposes in archeology, biology, sociology,

  → minimum spanning tree

  the points ----- vertices of a graph

  connections ---- the graph’s edges

  connection costs ---- the edge weights

Minimum Spanning Tree

- ***Problem:***
  - ***MST problem:*** *Given a connected, undirected, weighted graph G= (V, E), find a minimum spanning tree for it.*
    - *Compute MST through Brute Force?*
      - *Brute force*

        - generate all possible spanning trees for the given graph.

        - find the one with minimum total weight.
      - *Feasibility of Brute force*

        - Possible too many trees (exponential for dense graphs)
    - *Kruskal: 1956, Prim: 1957*

Minimum Spanning Tree

Prim

Idea of Prim

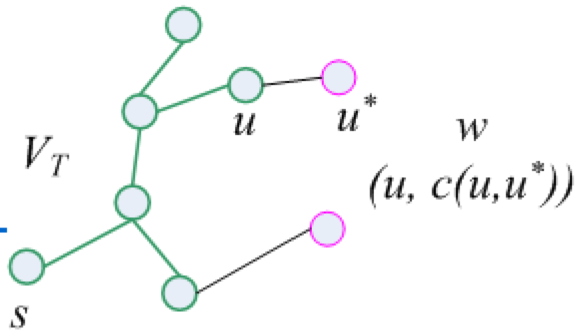
- *initial subtree T_0 consists of a single vertex selected arbitrarily from set V.*
- *a sequence of expanding subtrees, T_1, $T_{2\ ...}$ --- one vertex/edge at a time*
  - *$T_{i\text{-}1} \rightarrow T_i$ each stage*
  - *expands the current tree in the greedy*
- *On each iteration, attaching to tree the nearest vertex not in that tree.*
  - *the nearest vertex: a vertex not in the tree, and connected to a vertex in the tree by an edge of the smallest weight.*
- *stops after all the graph's vertices have been included in the spanning tree*
- *the algorithm expands a tree by exactly one vertex on each of its iterations, the total number of such iterations is n − 1, where n is the number of vertices in the graph.*
- *The tree generated by the algorithm is obtained as the set of edges used for the tree expansions.*

Minimum Spanning Tree

Prim -I

```
ALGORITHM Prim(G)
//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = (V, E)$
//Output: $E_T$, the set of edges composing a minimum spanning tree of
    $G$
$V_T \leftarrow \{v_0\}$    //$v_0$ can be arbitrarily selected
$E_T \leftarrow \phi$
for $i \leftarrow 1$ to $|V|-1$ do
    find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges $(v,
    u)$ such that $v$ is in $V_T$ and $u$ is in $V-V_T$
    $V_T \leftarrow V_T \cup \{u^*\}$
    $E_T \leftarrow E_T \cup \{e^*\}$
return $E_T$
```

Minimum Spanning Tree



- *Idea of Prim ('cont)*
  - *How to find the minimum weight edge that connecting a vertex in tree T_{i-1} to a vertex not yet in the tree ?*

    → attaching two labels to a vertex:

    - the name of the nearest tree vertex

    - the length (the weight) of the corresponding edge
  - *finding the next vertex to be added to the current tree $T_i = (V_T, E_T)$*

    ↔ finding a vertex with the smallest distance label in the set $V - V_T$

    $u \in V_T$, $u* \in V\text{-}V_T$, $\min\{ c[u, u*] \}$
  - *After we have identified a vertex u^* , make some modification to u's neighbors*
    - *Move u^* from the set $V - V_T$ to the set of tree vertices V_T .*
    - *For each remaining vertex w in $V - V_T$ that is connected to $u*$ by a shorter edge than the u's current distance label, update its labels by $u*$ and the weight of the edge between $u*$ and u, respectively.*

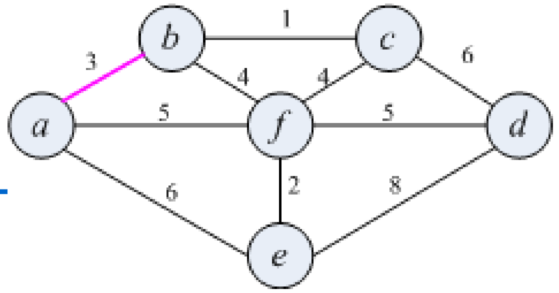
Minimum Spanning Tree

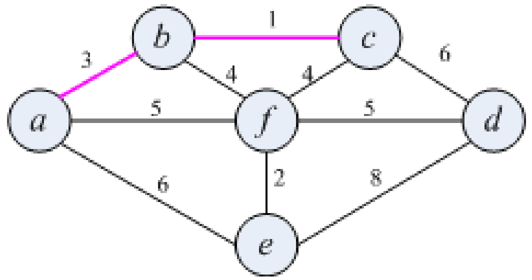
- **Prim is greedy**
  - *The choice of edges added to current subtree satisfying the three properties of greedy algorithms.*
    - Feasible, *each edge added to the tree does not result in a cycle, guarantee that the final ET is a spanning tree*
    - Local optimal, *each edge selected to the tree is always the one with minimum weight among all the edges crossing V_T and $V-V_T$*
    - *Irrevocable, once an edge is added to the tree, it is not removed in subsequent steps.*

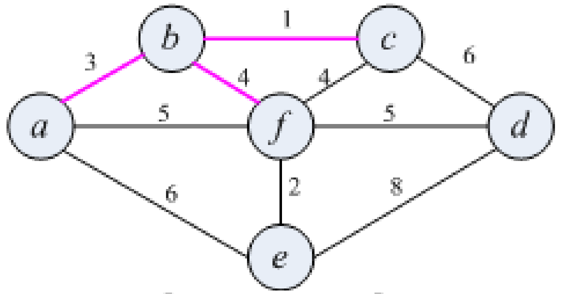
Minimum Spanning Tree

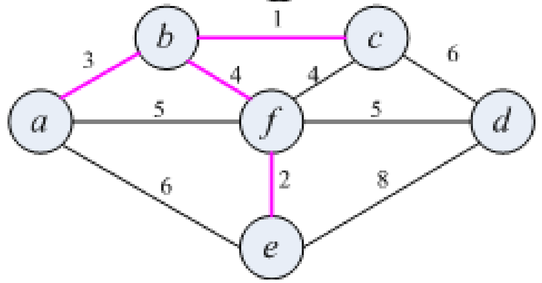
- *Example : Prim*

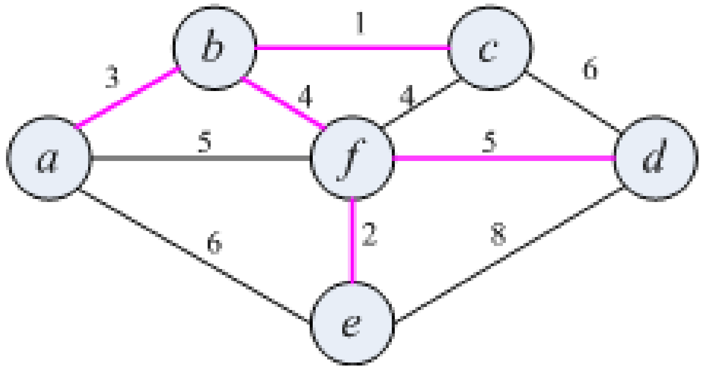
Tree vertices	Remaining vertices
a(-,0)	b(a,3) c(-,∞) d(-, ∞) e(a,6) f(a,5)
b(a,3)	c(b,1) d(-, ∞) e(a,6) f(b,4)
c(b,1)	d(c, 6) e(a,6) f(b,4)
f(b,4)	d(f, 5) e(f,2)
e(f,2)	d(f, 5)
d(f, 5)	











Minimum Spanning Tree

Kruskal's algorithm

Idea of Kruskal's

- *looks at a minimum spanning tree of a weighted connected graph G =(V,E) as an acyclic subgraph with |V| − 1 edges for which the sum of the edge weights is the smallest.*
- *sorting the graph's edges in nondecreasing order of their weights.*
- *starting with the empty subgraph, it scans this sorted list,*
- *adding the next edge on the list to the current subgraph if such an inclusion does not create a cycle and simply skipping the edge otherwise.*
  - need efficient way of detecting/avoiding cycles
- *algorithm stops when all vertices are included*

Minimum Spanning Tree

Kruskal's Algorithm (Advanced Part)

- *Some reviews:*
- *Kruskal's algorithm has to check whether the addition of the next edge to the edges already selected would create a cycle.*
- *a new cycle is created ↔ the new edge connects two vertices already connected by a path, ↔ the two vertices belong to the same connected component (Figure 9.6).*
- *Note also that each connected component of a subgraph generated by Kruskal's algorithm is a tree because it has no cycles.*

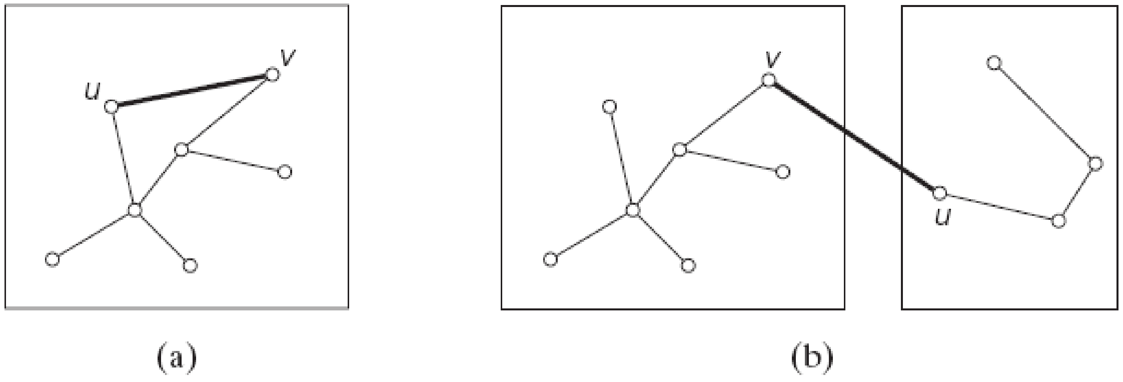


FIGURE 9.6 New edge connecting two vertices may (a) or may not (b) create a cycle.

Minimum Spanning Tree

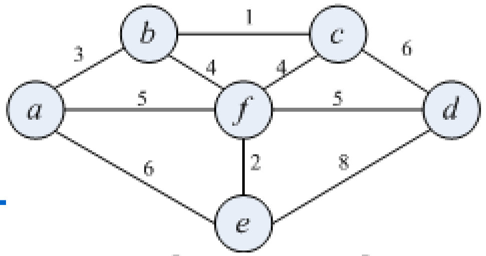
Kruskal’s Algorithm (Advanced Part)

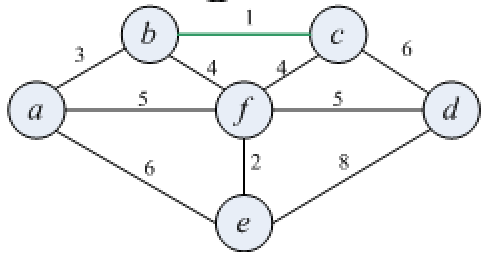
- *Kruskal’s alg. with a slightly different interpretation.*
- *consider the algorithm’s operations as a progression through a series of forests containing all the vertices of a given graph and some of its edges.*
- *The initial forest consists of |V | trivial trees, each comprising a single vertex of the graph. The final forest consists of a single tree, MST*
- *On each iteration, the algorithm takes the next edge (u, v) from the sorted list of the graph’s edges,*
- *finds the trees containing the vertices u and v, and, if these trees are not the same, unites them in a larger tree by adding the edge (u, v).*
- *union-find algorithms..*

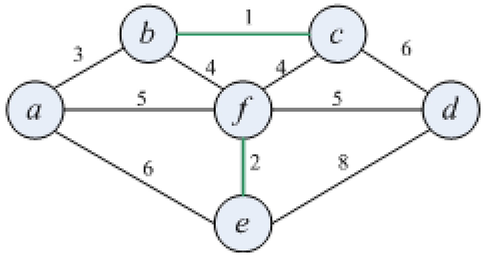
Minimum Spanning Tree

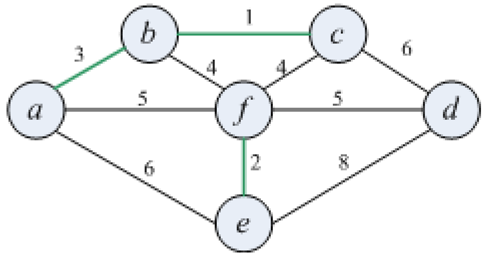
Example: Kruskal's

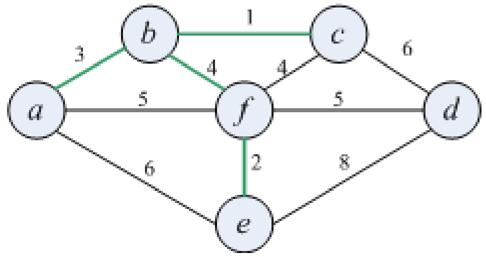
Tree edges	Sorted list of edges									
	bc	ef	ab	bf	cf	af	df	ae	cd	de
	1	2	3	4	4	5	5	6	6	8
bc										
1										
ef										
2										
ab										
3										
bf										
4										
df										
5										

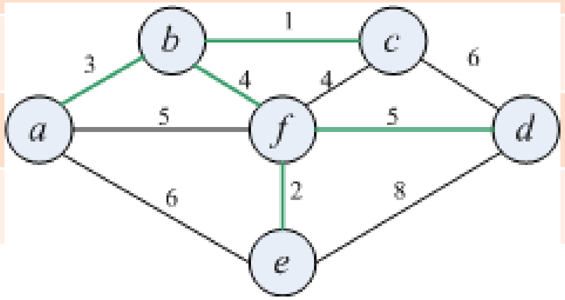




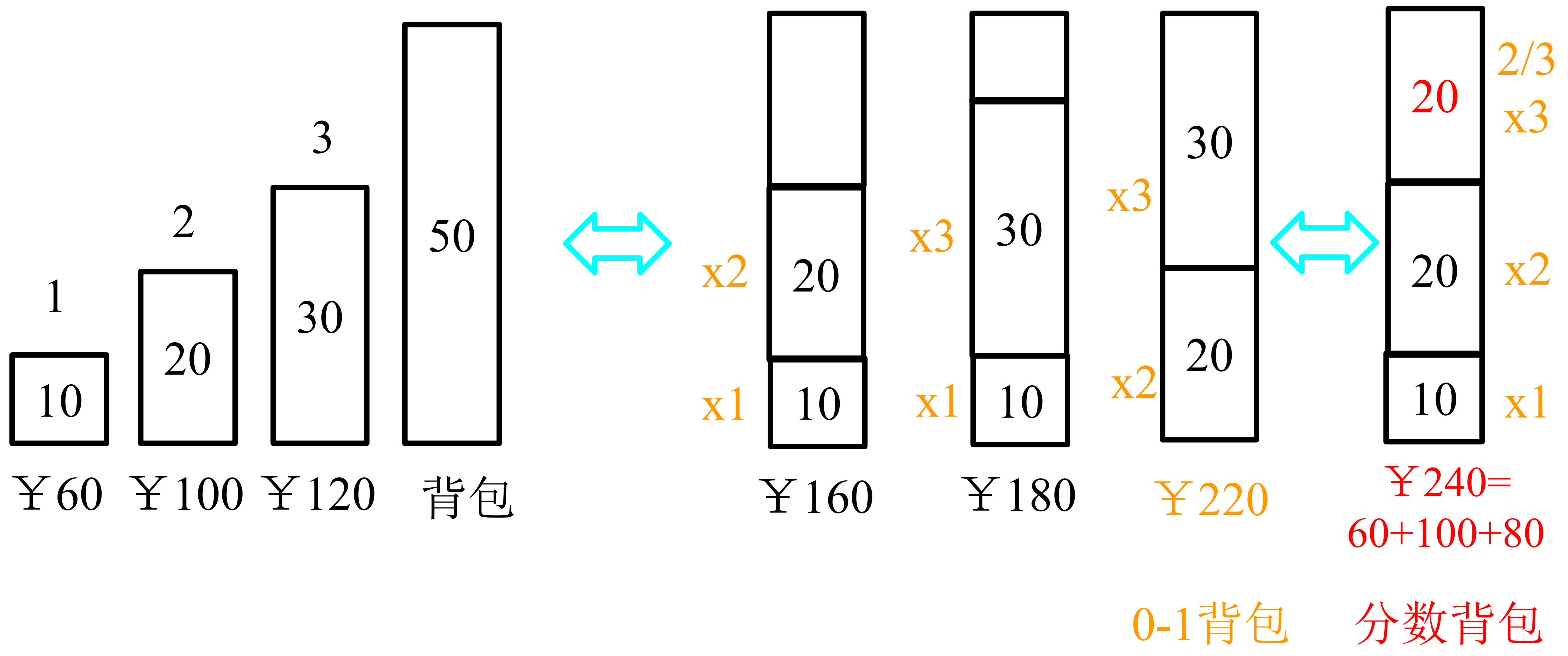








0-1背包 v.s. 分数背包



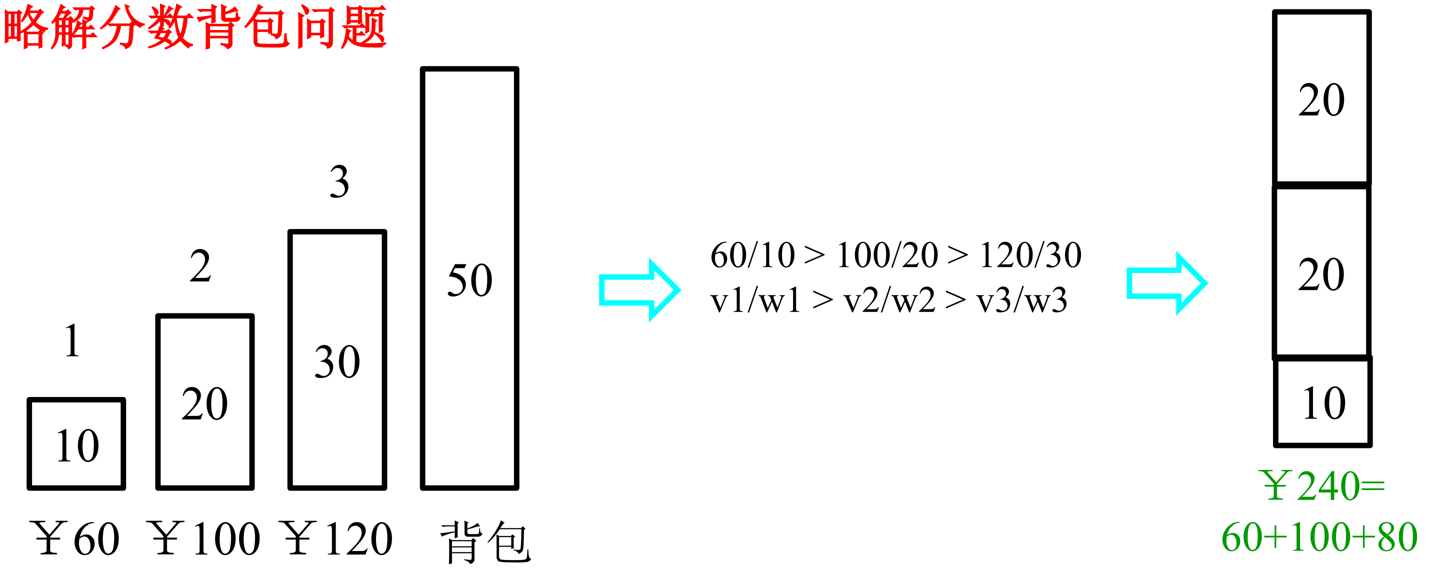
对于0-1背包问题，贪心选择不能得到最优解，但是动态规划可以

是因为在这种情况下，它无法保证最终能将背包装满，部分闲置的背包空间使每公斤背包空间的价值降低了。

解的限制，影响了算法的选择，demo

贪心算法 和 动态规划

- 贪心策略解分数背包问题



- 动态规划算法可以有效地解0-1背包问题

处理0-1背包问题时，应比较**选择该物品和不选择该物品所导致的最终方案**，然后再作出最好选择。

最终方案是由许多**重叠的子问题的解构成**。这正是该问题可用动态规划算法求解的另一重要特征。

		0	w-w_i	……	j	W
	0	0	0		0	0
w_i v_i	i-1	0	$V[i$-1, w-$w_i]$		$V[i$-1, $w]$	
	i	0	0		$V[i,w]$	
		0				
	n	0				**Target**

DP v.s. Greedy Algorithms

贪心选择性质

所谓贪心选择性质是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。这是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。

	动态规划（DP）	贪心算法（Greedy Method）
子问题上	每步所做的选择往往依赖于子问题的解，只有在解出相关子问题后才能作出选择	仅在**当前状态**下作出最好选择，即局部最优选择，然后再去作出这个选择后产生的相应的子问题，不依赖于子问题的解
求解方式	通常以自底向上的方式解各子问题	通常以**自顶向下**的方式进行，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题简化为规模更小的子问题

DP v.s. Greedy Algorithms

- 最优子结构性质：与动态规划的相似性

原问题的最优解包含子问题的最优解，即问题具有最优子结构的性质。

在背包问题中，第一次选择单位质量上价值最大的货物，它是第一个子问题的最优解，第二次选择剩下的货物中单位重量上价值最大的货物，同样是第二个子问题的最优解，依次类推。