Last Section

Divide and Conquer

- MINMAX:

  2n – 2 vs. 3n/2 – 2
- 二分搜索:

  算法BINARYSEARCHREC在n 个元素组成的已排序数组中搜索某个元素所执行的元素比较次数不超过 $\lfloor \log n \rfloor + 1$;

  非递归;
- 合并排序： nlogn
- 寻找第 k 小元素：20cn
- 划分算法与快速排序：n-1;n(n-1)/2

分治

- 大整数乘法： $n^{\log 3} \approx n^{1.59}$
- 矩阵乘法与STRASSEN 算法：$n^{\log 7} \approx n^{2.81}$
- 最近点对：nlgn

减 治

- 插入排序：n^2, n, $n^2/4$
- 快速排序+插入排序
- 拓扑排序：减一
- 生成排列+ Johnson-Trotter
- 生成子集+比特串方法

- 假币问题
- 俄式乘法
- 约瑟夫斯问题

- 欧几里德算法
- 插值查找
- 二叉查找树

生成子集

- 生成一个集合$A=\{a_1,\ldots,a_n\}$的所有2^n个子集。
- 减一思想

  集合$A=\{a_1,\ldots,a_n\}$的所有子集可以分为两组：

  不包含a_n的子集和那些包含a_n的子集。

  前一组其实就是$\{a_1,\ldots,a_{n-1}\}$的所有子集， 而后一组中的每一个元素都可以通过把a_n添加到$\{a_1,\ldots,a_{n-1}\}$的一个子集中来获得。

  因此，一旦我们得到了$\{a_1,\ldots,a_{n-1}\}$所有子集的列表，我们可以把列表中的每一个元素加上a_n， 再把它们添加到列表中，以得到$\{a_1,\ldots,a_n\}$的所有子集。

生成子集

n	子集
0	Φ
1	Φ {a1}
2	Φ{a1} {a2} {a1,a2}
3	Φ{a1} {a2} {a1,a2} {a3} {a1,a3} {a2,a3} {a1,a2,a3}

简便的比特串方法

- 基于这样一种关系：

  n个元素集合$A=\{a_1,\ldots,a_n\}$的所有2^n 个子集和长度为n的所有2^n 个比特串B之间的一一对应关系。

- 建立这样一种对应关系的最简单方法是为每一个子集制定一个比特串：

  如果a_i属于该子集的话，$b_i=1$, 如果a_i 不属于该子集的话，$b_i=0$。

  例：110表示$\{a_1, a_2\}$.

比特串方法

- 对于n=3的情况， 有：

比特串

000　001　010　011　100　101　110　111

子集

Φ　$\{a_3\}$　$\{a_2\}$　$\{a_2,a_3\}$　$\{a_1\}$　$\{a_1,a_3\}$　$\{a_1,a_2\}$　$\{a1,a_2,a_3\}$

Decrease by a constant factor

- 减常因子是减治方法的第二种主要变种：
  在算法的每次迭代中，总是从实例的规模中减去一个相同的常数因子。（在大多数应用中，这个常数因子等于二）

  —折半查找

假币问题

- 最能体现减常因子的Fake-coin版本
  - 天平，n 枚，折半分组

  （我们假设假币较轻）

  - 奇数，偶数

- 当n>1时，W(n)=W($\lfloor n/2 \rfloor$)+1, W(1) = 0
  - logn

    - 可否更高效？

俄式乘法

计算两个正整数n和m的乘积

- 若以n值为实例规模的度量：

  n*m=(n/2)*2m　　n为偶数

  n*m=((n-1)/2)*2m+m　　n为奇数

  以 1*m=m 为算法停止条件

  例：

- Why “Russian peasant” ?

俄式乘法

计算两个正整数n和m的乘积

- 若以n值为实例规模的度量：

  n*m=(n/2)*2m　　n为偶数

  n*m=((n-1)/2)*2m+m　　n为奇数

  以 1*m=m 为算法停止条件

  例：

- Why “Russian peasant” ?　　X2,/2,+ !

约瑟夫斯问题

- 求幸存者的号码 J（n）
- J(1)=1
- n为偶数：　　J(2k)=2J(k)-1
  (初始=新*2-1，且幸存者将保持这种关系)
- n为奇数:　　J(2k+1)=2J(k)+1
- 二进制表示n,左循环移位一次

Variable size decrease

- **减可变规模**是减治方法的第三种主要变种：
  算法在每次迭代时，规模减小的模式都和另一次迭代是不同的。

- Euclid's gcd Algo.
- Selection problem (The k^{th} order statistic)
  (The k^{th} smallest element)

插值查找

- Interpolation search （sorted array）

$$x = l + \left\lfloor \frac{(v - A[l])(r - l)}{A[r] - A[l]} \right\rfloor$$

- 查找n个随机键的列表的平均比较次数：loglogn+1

- 最差：*O*(n)

二叉查找树

- 是一种节点包含可排序项集合中元素的二叉树， 每个节点一个元素， 并使得 对于每个节点来说， 所有左子树的元素都小于树根节点的元素， 所有右子数的元素都大于树根节点的元素。

- 递归查找 v:

  把*v* 和该树的根K(r)进行比较:

  如果相等， 查找停止;

  如果不相等:

  当*v*<*K*[*r*] 时，在左子树中继续查找;

  当*v*>*K*[*r*]时， 在右子树中继续查找。

二叉查找树

- 在算法的每次迭代中， 查找一棵二叉查找树的问题， 简化为查找一棵更小的二叉查找树。

- 一棵查找树的规模的最佳量度就是树的高度； 显然， 在二叉树的查找中，从一次迭代到另一次迭代， 树的高度减少通常都不相同
  - 这是一个很好的关于减可变规模算法的例子。

二叉查找树

Worst case: 树是严格歪斜的（severely skewed)

n次比较，Θ(n)

Average case: 约2ln(n) 次比较≈1.39logn

Θ(logn)

（由n个随机键构造起来的二叉查找树）

Transform and Conquer

变治

Transform and Conquer

- 根据对问题实例的变换方式，变治思想有3种主要类型：

1. 变换为同样问题的一个更简单或者更方便 的实例—**实例化简(Instance simplification)**。
2. 变换为同样实例的不同表现—**改变表现(Representation Change).**
3. 变换为另一个问题的实例， 这种问题的算法是已知的—**问题规约(Problem reduction).**

实例化简_预排序

- 如果列表是有序的，许多问题更容易求解。
- 例：检验数组中元素的惟一性

UniqueElements(A[1,…n])
//输出：如果*A*之元素全部惟一，返回“true”, 否则返回“false”

1. **for** i←1 **to** n-1 **do**
2. **for** j← i+1 **to** n **do**
3. **if** A[i]=A[j] **return false**
4. **return true**

n(n-1)/2

检验数组中元素的惟一性

先对数组排序，然后只检查它的连续元素：如果该数组有相等的元素，则一定有一对元素是相互紧挨着的，反之亦然。

PresortElementUniqueness(*A*[1..*n*])
//先对数组排序来解元素惟一性问题
//输入：*n*个可排序元素构成的一个数组*A*[1..*n*]
//输出：如果*A*没有相等的元素，返回“true”, 否则返回”false”

1. 对数组A排序
2. **for** i← 1 to *n*-1 **do**
3. **if** *A*[*i*] = *A*[*i*+1] **return false**
4. **return true**

$T(n)=T_{sort}(n) + T_{scan}(n) \in \Theta(n\log n) + \Theta(n) = \Theta(n\log n)$

实例化简_预排序

例：模式计算

- 在给定的数字列表中最经常出现的一个数值称为模式（Mode）。（例：5，1，5，7，6，5，7，模式是5）
- 如果若干个不同的值都是最经常出现的，它们中的任何一个都可以看作是模。

模式计算

- 蛮力法：对列表进行扫描， 并计算它的所有不同值出现的频率。
- 可以在另两个列表中存储已经遇到的值和它们的出现频率。
- 每次迭代时，通过遍历储值辅助列表，使原始列表中的第*i*个元素和已遇到的数值进行比较。
- 如果碰到一个匹配数值，相应的储频辅助列表中该数值的出现频率加1；否则，将当前元素添加到辅助列表中，并把它的出现频率置为1。

模式计算

- 第 i 个元素与辅助列表中的 i-1 个元素比较
- 蛮力法的最坏情况比较次数是：

$$C(n)=\sum_{i=1}^{n}(i-1) \ =0+1+\ldots+(n-1)=(n-1)\,n/2$$

+求辅助列表中最大频率之 n-1 次

模式计算

- 预排序：

  -- 先对输入排序；之后所有相等的数值都会邻接在一起。

  -- 为了求出模式，需要做的全部工作就是求出在该有序数组中邻接次数最多的等值元素。

模式计算

PresortMode(*A*[1..*n*])
//输入：可排序元素构成的数组**A**[1..n]
//输出：该数组的模式

1. 对数组**A**排序
2. *i*←1 //当前一轮从位置*i*开始
3. *modefrequency* ←0 //目前为止求出的最高频率
4. **while** i≤n **do**
5. *length* ←1; *value*←A[i]
6. **while** *i*+*length*≤*n* **and** *A*[*i*+*length*]=*value*
7. *length* ← *length* +1
8. **if** *length* > *modefrequency*
9. *modefrequency*←*length*; *modevalue* ←*value*
10. *i*←*i*+*length*
11. **return** *modevalue*

模式计算

- 模式计算之预排序算法：

  排序时间+线性时间

  nlogn

实例化简_高斯消去法

- 高斯消去法的思路是把*n*个线性方程构成的*n* 元联立方程组变换为一个等价的方程组（即它的解和原来的方程组相同）。

  Ax=b

- 该方程组有着一个上三角形的系数矩阵，这种矩阵的主对角线下方元素全部为0：

实例化简_高斯消去法

- 用第二个方程和第一个方程乘以a_{21}/a_{11}之后的差来替代第二个方程， 以得到一个x_1系数为0的方程。
- 对第三个、第四个、直到最后第n个方程，分别用a_{31}/a_{11}, a_{41}/a_{11}…,a_{n1}/a_{11}, 作为第一个方程的乘数，重复同样的做法，使得第一个方程以后的所有x1的系数都为0。
- 然后，从第二个方程以后的每个方程中减去第二个方程的一个恰当的倍数，从而除去了它们全部x_2的系数。
- 对前面*n*-1个变量中的每一个都重复这个过程，最终会生成一个具有上三角系数矩阵的方程组。

高斯消去法

GaussElimination(A[1..n,1..n],b[1..n])

//输入：矩阵A[1..n,1..n]和列向量b[1..n]

//输出：一个代替A的上三角形等价矩阵，等式右边的值位于第（n+1）列中

```
1. for i← 1 to n do A[i,n+1] ←b[i]  //扩展该矩阵
2. for i←1 to n-1 do                //Eliminate/Diag
3.     for j←i+1 to n do            //Row
4.         for k←i to n+1 do        //Col
5.             A[ j,k] ←A[ j,k]-A[ i,k]*A[ j,i]/A[ i,i]
```

高斯消去法_Partial pivoting

- 两个问题：
  - A[i,i]=0 ；
  - A[i,i]可能会非常小，比例因子A[j,i]/A[i,i]非常大，以至于A[j,k]的新值会因为舍入而严重失真。

- 解决办法— Partial pivoting

  每次都去找第i列系数的绝对值最大的行，然后把它作为第i次迭代的基点。

  保证比例因子的绝对值永远不会大于1。

高斯消去法_Partial pivoting

BetterGaussElimination(A[1..n,1..n],b[1..n])
//输入：矩阵*A*[1..*n*,1..*n*]和列向量*b*[1..*n*]
//输出：一个代替 A 的上三角行等价矩阵

```
1.  for i←1 to n do A[i,n+1] ←b[i ]
2.  for i←1 to n-1 do                          // E(col)
3.      pivotrow ← i
4.      for j←i+1 to n do                      // Row
5.          if |A[j,i ]|>|A[pivotrow,i ]| then pivotrow←j
6.      for k←i to n+1 do                      // Col
7.          swap(A[i,k],A[pivotrow,k])
8.      for j←i+1 to n do                      // Row
9.          temp ←A[j,i ]/A[i,i ]
10.         for k←i to n+1 do                  // Col
11.             A[j,k] ←A[j,k]-A[i,k]*temp
```

高斯消去法_Partial pivoting

- 最内层循环只有一次乘法

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=i}^{n+1} 1$$

- $\approx n^3/3$

LU 分解

为解 Ax=b

- L：由主对角线上的1和在高斯消去过程中,行的乘数所构成的下三角矩阵。
- U：高斯消去过程后的上三角矩阵。
- 有：LU=A
- 因：Ax=b，有：LUx=b.
- 设：y=Ux, 则Ly=b,解之得y
- 再解: Ux=y,得到 x.

*对任意列向量 b 通用

矩阵的逆与LU分解

- $AA^{-1}=I$
- 我们可以通过解n个具有相同系数矩阵A 的线性方程组来求出这些未知数，未知向量 x^j 是逆矩阵的第 j 列，右边的向量 e^j 是单位矩阵的第j 列（1≤j≤n）：

$$Ax^j = e^j$$

- 可以应用高斯消去法（LU分解）来解这些方程组。

实例化简_AVL树

- 二叉查找树：
  二叉树节点所包含的元素来自可排序项的集合，每个节点一个元素，使得所有左子树中的元素都小于树根节点的元素，而所有右子树中的元素都大于它。（把一个集合变换为一棵 二叉查找树，是改变表现技术的一个实例）

- 递归查找 v:
  把*v* 和该树的根K(r)进行比较:
  如果相等， 查找停止;
  如果不相等:
  当*v*<*K*[*r*] 时，在左子树中继续查找;
  当*v*>*K*[*r*]时， 在右子树中继续查找。

二叉查找树

- 在算法的每次迭代中， 查找一棵二叉查找树的问题， 简化为查找一棵更小的二叉查找树。

- 一棵查找树的规模的最佳量度就是树的高度； 显然， 在二叉树的查找中，从一次迭代到另一次迭代， 树的高度减少通常都不相同
  - 这是一个很好的关于减可变规模算法的例子。

二叉查找树的变治应用

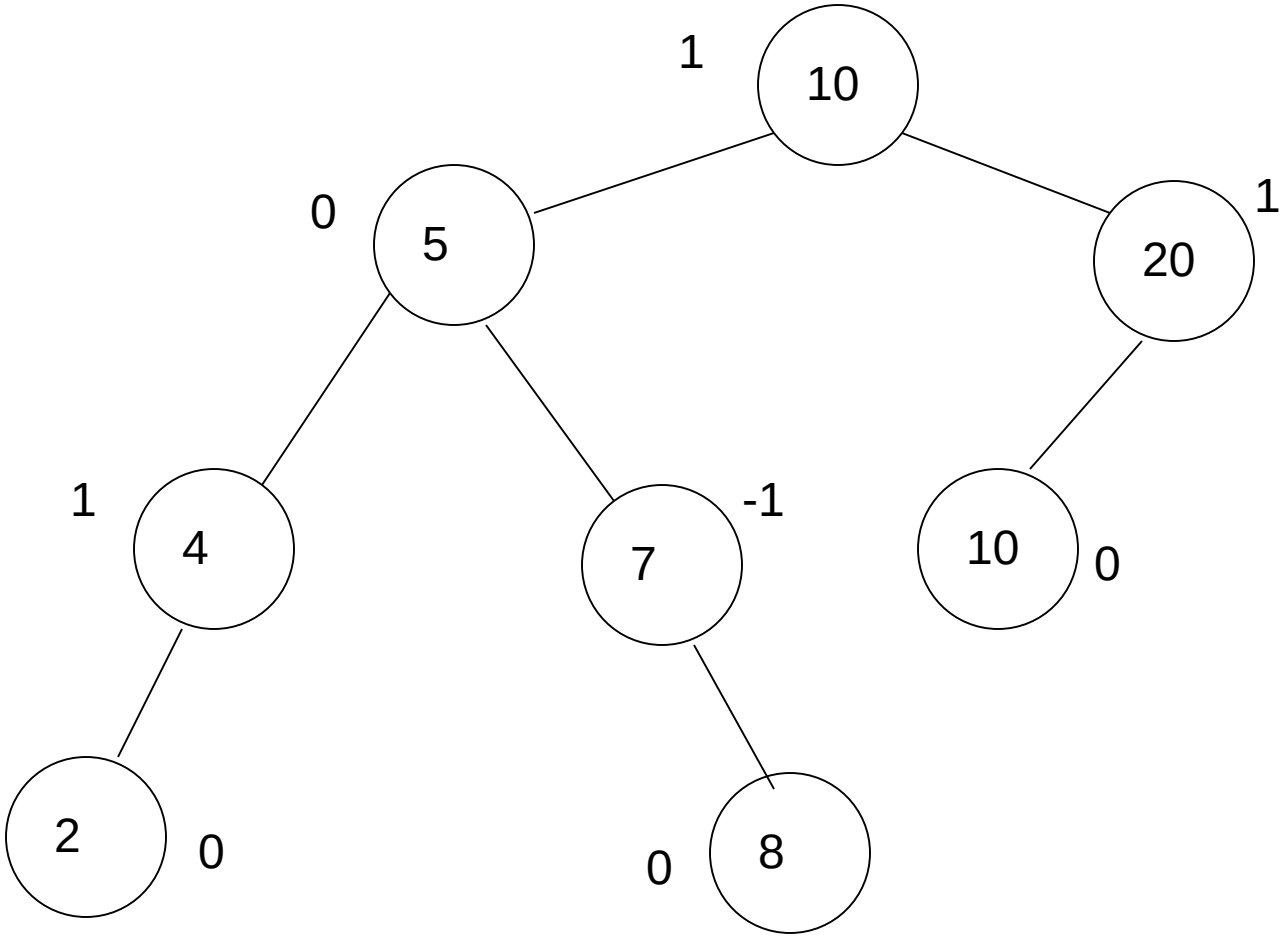
- 赢得了查找、插入和删除的时间效率， 这些操作都属于Θ(lgn) 。 但这仅仅在平均情况下成立，在最差情况下，这些操作属于Θ(n) ， 因为这种树可能会退化成一种严重不平衡的树， 树的高度等于n-1。

- 两种方案：
  - 一棵*AVL*树要求它的每个节点的左右子树的高度差不能超过1。（实例化简：红黑、分裂 ）
  - **2-3**树、**2-3-4**树允许一棵查找树的单个节点中不止包含一个元素 。（改变表现 ）

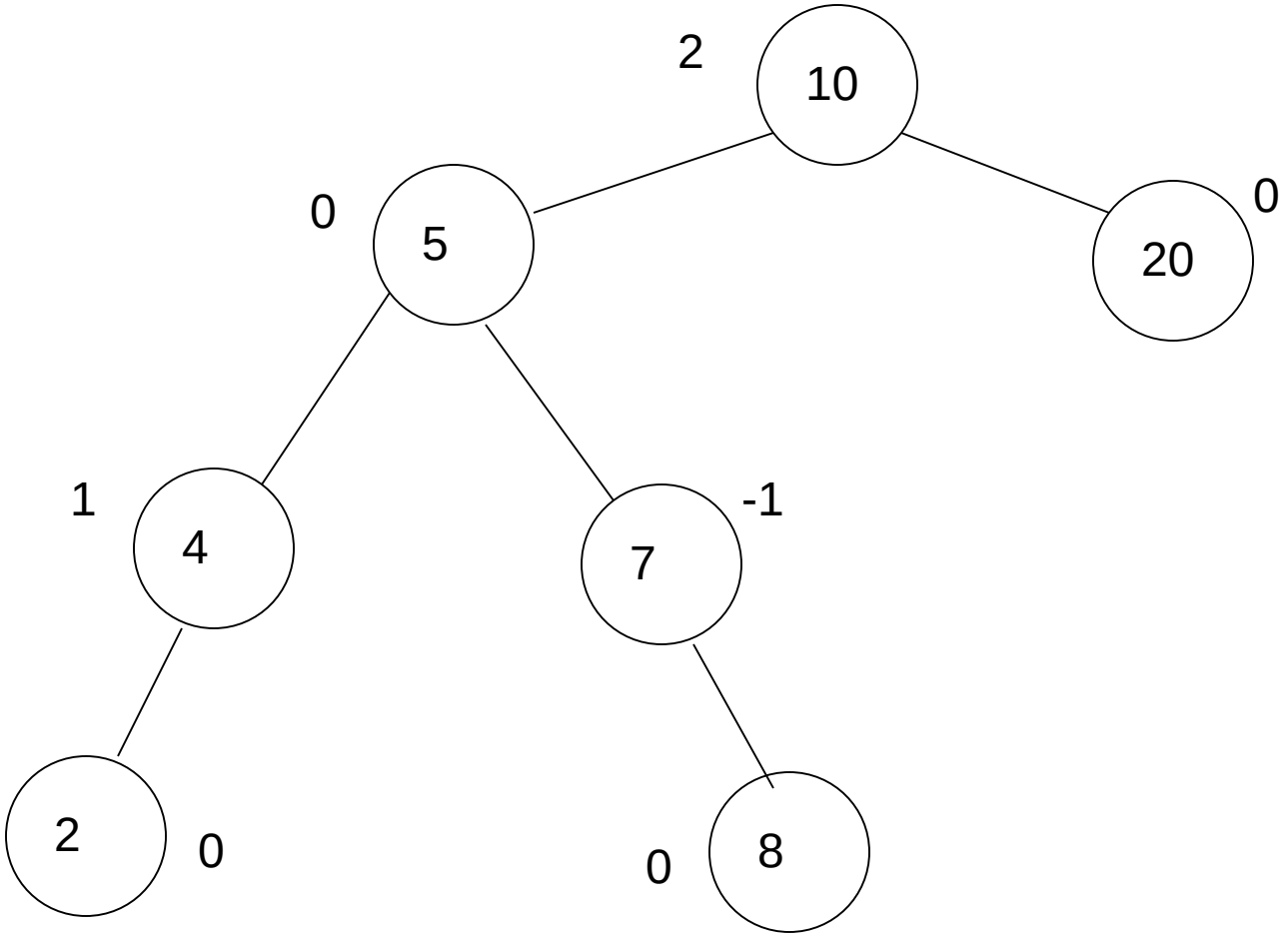
AVL 树

- **定义**　　一棵**AVL树**是一棵二叉查找树，其中每个节点的**平衡因子**定义为该节点左子树和右子树的高度差，这个平衡因子要么为0，要么为+1或者-1。

AVL树



非AVL树



AVL树的旋转

- 如果插入一个新节点使得一颗AVL树失去了平衡，则用旋转对此AVL树进行变换。

- AVL树的旋转：以某节点为根的子树的一个本地变换，其平衡因子为2或-2；如有若干此种节点，则以最靠近新插入节点的节点为根。

- 有四种类型的旋转：
  - 右单转
  - 左单转
  - 左右双转
  - 右左双转

*例：为列表4，7，9，2，1，3，8 构造一AVL树

AVL树的效率

- 所有包含n个节点的AVL树的高度h 都满足这个不等式：

  $\lfloor \log_2 n \rfloor \le h < 1.4405\log_2(n+2) - 1.3277$

- 在最差情况下，查找和插入操作的效率属于Θ(lgn)。

- 对于一棵针对随机键的列表构造的AVL树 来说，得到它的平均高度的精确公式被证明是有难度的；实验表明，除非n 比较小，否则，这个高度大概是$1.01\log_2 n + 0.1$.

  因此，在平均情况下，查找一棵AVL树需要的比较次数和用折半查找查找一个有序数组是几乎相同的

改变表现_ 2-3 树

- 平衡一棵查找树的第二种思路是允许一个节点不止包含一个键。

- 2-3树是一种可以包含两种类型节点的树：
  - 2节点
  - 3节点。

2-3 树

- 一个2节点只包含一个键K和两个子女（子节点）：
  - 左子节点作为一棵所有键 都小于K 的子树的根
  - 右子节点作为一棵所有键都大于K的子树 的根

    （一个2节点和一棵经典二叉查找树的节点类型是相同的）
- 一个3节点包含两个有序的键K1和K2（K1<K2）并且有3个子节点：
  - 最左边的子节点作为键值小于K1的子树的根
  - 中间的子节点作为键值位于K1和K2之间的子树的根
  - 最右边的子节点作为键值大于K2的子树的根。
- 2-3 树的最后一个要求是，树中的所有叶子必须位于同一层：
  - 2-3 树总是高度平衡的

2-3 树的查找

- 根是2节点：
  - 等同于二叉查找树
  - 比较于根键值：停止，左、右子树继续
- 根是3节点：
  - 不超过两次比较
  - 比较于根键值：停止，左、中、右子树继续

2-3 树的插入

- 如果插入位置的叶子是一个2节点
  根据K 是小于还是大于节点中原来的键，我们把K作为第一个键或者第二个键进行插入。

- 如果叶子是一个3节点

把叶子分裂成两个节点：

-- 三个键（两个原来的键和一个新键）中最小的放到第一个叶子里

-- 最大的放到第二个叶子中

-- 中间的键提升到原来叶子的父节点中去（如果这个叶子恰好是树的根，我们就创建一个新的根来接纳这个中间键）。

*中间键提升到父节点中去可能会导致父节点的溢出（如果它是一个3节点），并且因此会导致沿着叶子的祖先链条发生多个节点的分裂。

*例：为列表9，5，8，3，2，4，7 构造一 2-3树

2-3 树的效率

- 一个具有最少节点的高度为h的2-3树是一颗全部由2节点构成的满树。
- 对于任何包含n个节点、高度为h的2-3树，有：

$$n \geq 1+2+\ldots+2^h = 2^{h+1}-1$$

则 $h \leq \log_2(n+1)-1$

- 一个具有最多节点的高度为h的2-3树是一棵全部由3节点构成的满树，每个节点都包含两个键和三个子女。所以，对于任何n个节点的2-3树，有：

$$n \leq 2\cdot 1 + 2\cdot 3 + \ldots + 2\cdot 3^h = 2(1 + 3 + \ldots + 3^h) = 3^{h+1} - 1$$

则 $h \geq \log_3(n+1)-1$

2-3 树的效率

- 高度h的上下界是：

  $\log_3(n+1)-1 \leqslant h \leqslant \log_2(n+1)-1$。

- 即，无论在最差情况还是在平均情况，查找、插入和删除的时间效率都属于

  $\Theta(\log n)$

堆和堆排序

- 变治之

  改变表现

- 序列之于

  二叉树

- 序列之于

  二叉堆

霍纳法则

- 问题：
  针对一个给定的x 的多项式

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

  求值

- 霍纳法则是一个很好的改变表现技术的例子。
  它不断地把x作为公因子从降次以后的剩余多项式中提取出来：

$$p(x) = (\ldots(a_n x + a_{n-1})x + ..)x + a_0$$

霍纳法则

对于多项式 $p(x) = 2x^4 - x^3 + 3x^2 + x - 5$,

有：

$$
\begin{aligned}
p(x) &= 2x^4 - x^3 + 3x^2 + x - 5 \\
&= x（2x^3 - x^2 + 3x + 1）- 5 \\
&= x（x（2x^2 - x + 3）+1）-5 \\
&= x（x（x（2x-1）+3）+1）-5
\end{aligned}
$$

霍纳法则

- 用一个两行的表来帮助计算：
  - 第一行包含了该多项式的系数
  - 第二行中，除了第一个单元用来存储a_n， 其他单元都用来存储中间结果
  - 用第二行的最后一个单元乘以x的值再加上第一行的下一个系数， 来算出表格下一个单元的值
  - 以这种方式算出的最后一个单元的值，就是该多项式的值。

*例：计算 $p(x) = 2x^4 - x^3 + 3x^2 + x - 5$ 在x=3时的值

系数	2	-1	3	1	-5
X=3	2	3*2+(-1)=5	3*5+3=18	3*18+1=55	3*55-5=160

霍纳法则

Horner(*P*[0..*n*],*x*)
//用霍纳法则求一个多项式在一个给定点的值
//输入：一个*n*次多项式的系数数组*P*[0..*n*](从低到高存储)，
　　以及一个数字*x*
//输出：多项式在x点的值

1. *p*←*P*[*n*]
2. **for** *i*←*n*-1 **downto** 0 **do**
3. 　　*p*←*x***p*+*P*[*i*]
4. **return** *p*

乘法和加法次数均为 n

二进制幂

- 霍纳法则计算a^n时，它退化成了一种对a 自乘的蛮力算法，以及一些无用的加法。

- 两种基于改变表现思想的计算a^n 的算法：
  - 从左至右处理二进制串（n的二进制表示）
  - 从右至左处理

二进制幂

- 设$n = b_I \ldots b_i \ldots b_0$是在二进制系统中，表示一个正整数n 的比特串，则可以通过以下多项式的值来计算n：

$$p(x) = b_I\, x^I + \ldots + b_i\, x^i + \ldots + b_0$$

其中x = 2。

- 应用霍纳法则计算$p(2)$

$p \leftarrow 1$ // n>=1, 第一个数字总是1

for $i \leftarrow$ I-1 **downto** 0 **do**

$p \leftarrow 2\,p + b_i$

二进制幂

- $a^{n} = a^{p(2)}$:

  $a^{p} \leftarrow a^{1}$

  for $i \leftarrow$ I-1 **downto** 0 **do**

  $a^{p} \leftarrow a^{2p + bi}$

- 另

$$a^{2p+b_i} = a^{2p} \cdot a^{b_i} = (a^{p})^{2} \cdot a^{b_i} = (a^{p})^{2} \text{ 如果} b_i = 0$$

$$= (a^{p})^{2} \cdot a \quad \text{如果} b_i = 1$$

二进制幂

LeftRightBinaryExponentiation(*a*,*b*(*n*))
//用从左至右二进制幂算法计算a^n
//输入：一个数字*a*和二进制位b_I,.., b_0 的列表*b*(*n*),
　　这些位来自于一个正整数n的二进制展开式
//输出：a^n 的值

1. *product* ←*a*
2. **for** *i*←*I*-1 **downto** 0 **do**
3. 　　*product* ←*product***product*
4. 　　**if** b_i = 1 **then** ***product*** ←***product**a**
5. **return** *product*

二进制幂

- 因为该算法在每次重复它惟一循环的时候都要做一到两次的乘法，所以它在计算a^n时，总的乘法次数$M(n)$是

$$b\text{-}1 \leqslant M(n) \leqslant 2(b-1)$$

b 是代表指数n 的比特串的长度

$$b-1 = \lfloor \log_2 n \rfloor$$

- Vs. 减半

问题规约（Reduction)

- P1 reduces to P2 which can be solved by Algo A
- Solve P2 with Algo A
- Solution of P2 transformed to P1

问题规约_lcm

- Lcm(24,60)=120

  24=2*2*2*3 质数因子

  60=2*2*3*5

  Lcm(24,60)=(2*2*3) *2*5
- 缺乏效率，并且需要一个连续质数的列表。
- 问题化简：

  lcm(*m*,*n*)和gcd(*m*,*n*)的积把*m* 和*n* 的每一个因子都恰好包含了一次，因此就简单地等于*m* 和*n* 的积。

$$\text{lcm}(m,n) = \frac{m \cdot n}{\gcd(m,n)}$$

问题规约

- 综合除法

  （$2x^4 - x^3 + 3x^2 + x - 5$）/（x-3）的商$2x^3 + 5x^2 + 18x + 55$；余数 160。

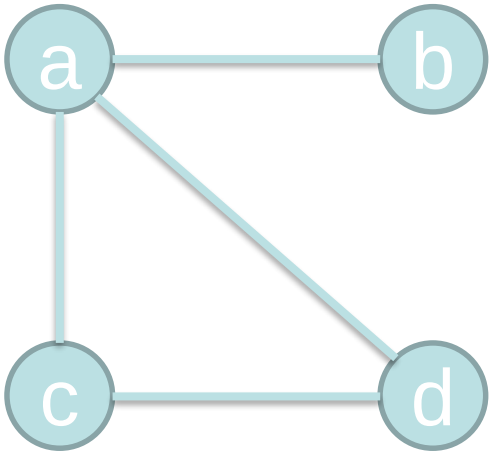
- 凸包：点的相对位置。

- 解析几何：几何-代数

计算图中的路径数量

- 从图（无向图或有向图）中第i个顶点到第j个顶点之间，长度为$k>0$的不同路径的数量等于A^k的第(i,j)个原素，其中，A是该图的邻接矩阵。

- 例：



$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} A & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$A^2 = \begin{bmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$$

规约为图问题

- 状态空间图：
  顶点，初始状态，目标状态
  边，状态之间的可能转变

- 一过河迷题版本：

规约为线性规划问题

- 线性规划问题是一个多变量线性函数的最优化问题，这些变量所要满足的一些约束是以线性等式或线性不等式的形式出现的。

$$
\begin{aligned}
&opt \quad z = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \\
&s.t. \\
&a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n = b_1 \\
&a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n = b_2 \\
&\ldots \\
&a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n = b_m
\end{aligned}
$$

规约为线性规划问题

- 线性规划问题是一个多变量线性函数的最优化问题，这些变量所要满足的一些约束是以线性等式或线性不等式的形式出现的。

- 背包问题：

  给定一个承重为W的背包和n个重量为$w_1,\ldots, w_n$、价值为$v_1,\ldots,v_n$的物品，求出这些物品中最有价值的一个子集，并且要能够装到背包中。

背包问题与线形规划

- 背包问题的连续版本可以表示为下面这个线性规划问题:

$$\max \sum_{j=1}^{n} v_j x_j$$

$$\text{S.T.} \quad \sum_{j=1}^{n} w_j x_j \le w$$

$$0 \le x_j \le 1,\ j = 1,\ldots,n$$

- 离散版本

$$x_j \in \{0,\ 1\},\ j = 1,\ldots,n$$

MST & Element Uniqueness

- MST 与 EU 哪一个可以规约为另一个问题？
- 规约的过程是怎样的?
- 哪一个更“容易”？

Dynamic Programming

动态规划

动态规划

- 动态规划是解决多阶段决策过程最优化的一种方法。

- 对于离散问题，解析数学无法施展，动态规划则成为一非常有效的工具。

- 两个弱点：
  1. 得出目标函数方程后，尚无统一的处理方法，必须根据具体问题的性质结合相应的数学技巧来求解；
  2. 维数障碍。

动态规划

- 动态规划模型的分类：
- 根据决策过程的时间参量是离散的还是连续的变量
  - 离散（多段）决策过程
  - 连续决策过程
- 根据决策过程的演变是确定性的还是随机性的
  - 确定性决策过程
  - 随机性决策过程
- 离散确定性
- 离散随机性
- 连续确定性
- 连续随机性

多阶段决策问题

- 由于它的特殊性，可将过程划分为若干互相联系的阶段；
- 在它的每一个阶段都需要作出决策，并且一个阶段的决策确定以后，常影响下一个阶段的决策，从而影响整个过程的活动路线；
- 各个阶段所确定的决策就构成一个决策序列，通常称为一个策略；
- 每一个阶段可供选择的决策往往不止一个，对应于一个策略就有确定的活动效果；
- 多阶段决策问题，就是要在允许选择的那些策略中间，选择一个最优策略，使在预定的标准下达到最好的效果。

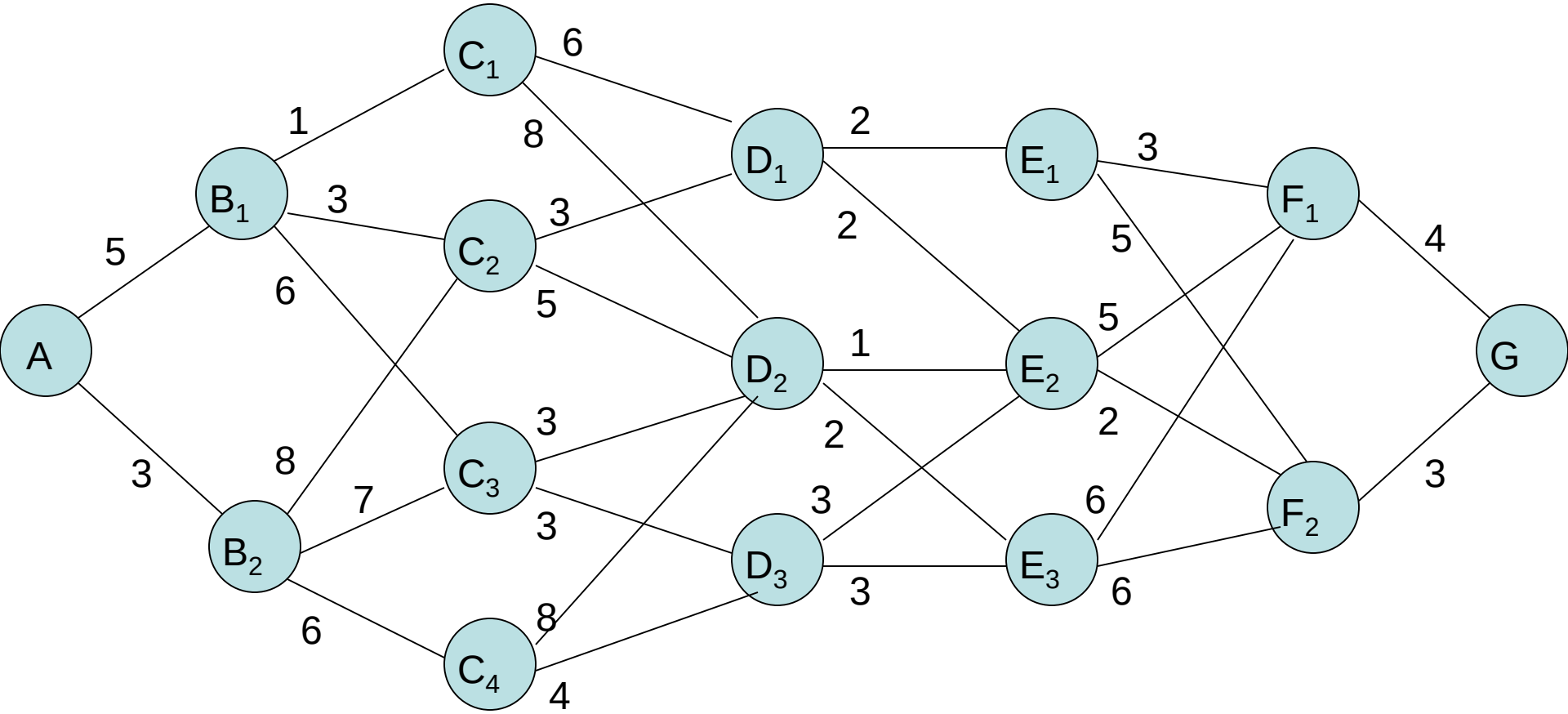
问题举例

例1　最短路线问题

给出一个线路网络， 从A点要铺设一条管道到G点，其两点之间连线上的数字表示两点间的距离；

要求选择一条由A到G的铺管线路，使总距离为最短。

最短路线问题


A
B1
B2
C1
C2
C3
C4
D1
D2
D3
E1
E2
E3
F1
F2
G
5
3
1
6
3
8
6
8
7
6
3
8
4
2
3
2
1
2
3
3
3
5
5
2
6
3
4
3
6

问题举例

例2 机器负荷分配问题

某种机器，可以在高低两种不同的负荷下进行生产。

在高负荷下进行生产时，产品的年产量s_1和投入生产的机器数量u_1的关系为

$$s_1 = g(u_1)$$

这时，机器的年折损率为a，即如果年初完好机器的数量为u，到年终时完好的机器就为au, $0<a<1$.

在低负荷下生产时，产品的年产量s_2和投入生产的机器数量u_2的关系为

$$s_2 = h(u_2)$$

相应的机器的年折损率为b, $0<b<1$.

假定开始生产时完好的机器数量为x_1。要求制定一个五年计划，在每年开始时，决定如何重新分配完好的机器在两种不同的负荷下生产的数量，使在五年内产品的总产量达到最高。

例3 部件的生产计划问题

- 某车间需要按月在月底供应一定数量的某种部件给总装车间。
- 由于生产条件的变化，该车间在各月份中，生产每单位这种部件所耗费的工时不同。
- 各月份的生产，除供应该月的需要外，余下部分可存入仓库备用。但因仓库容量的限制，库存部件的数量不能超过某一给定值H。
- 已知半年期间的各个月份的需求量，以及在这些月份生产该部件每单位数量所需工时数如表所示。
- 设仓库容量限制H= 9， 开始库存量为2，期末库存量为0。
- 要求制定一个半年的逐月生产计划，使在满足需要和库存量限制的条件下，生产这种部件的总耗费工时数达到最小。

月份k	0	1	2	3	4	5	6
需求量d_k	0	8	5	3	2	7	4
单位工时a_k	11	18	13	17	20	10	

问题举例

例4 不定步数的最短路线问题

给定M个点p_1，p_2，……，p_M，其中任意两点p_i和p_j (1≤i≤M，1≤j≤M)间的距离c_{ij}是已知的，从一点直达另一点称为一步。要求在不限定步数的条件下，找出由p_i到达p_M的最短路线。

动态规划的基本概念和基本方程

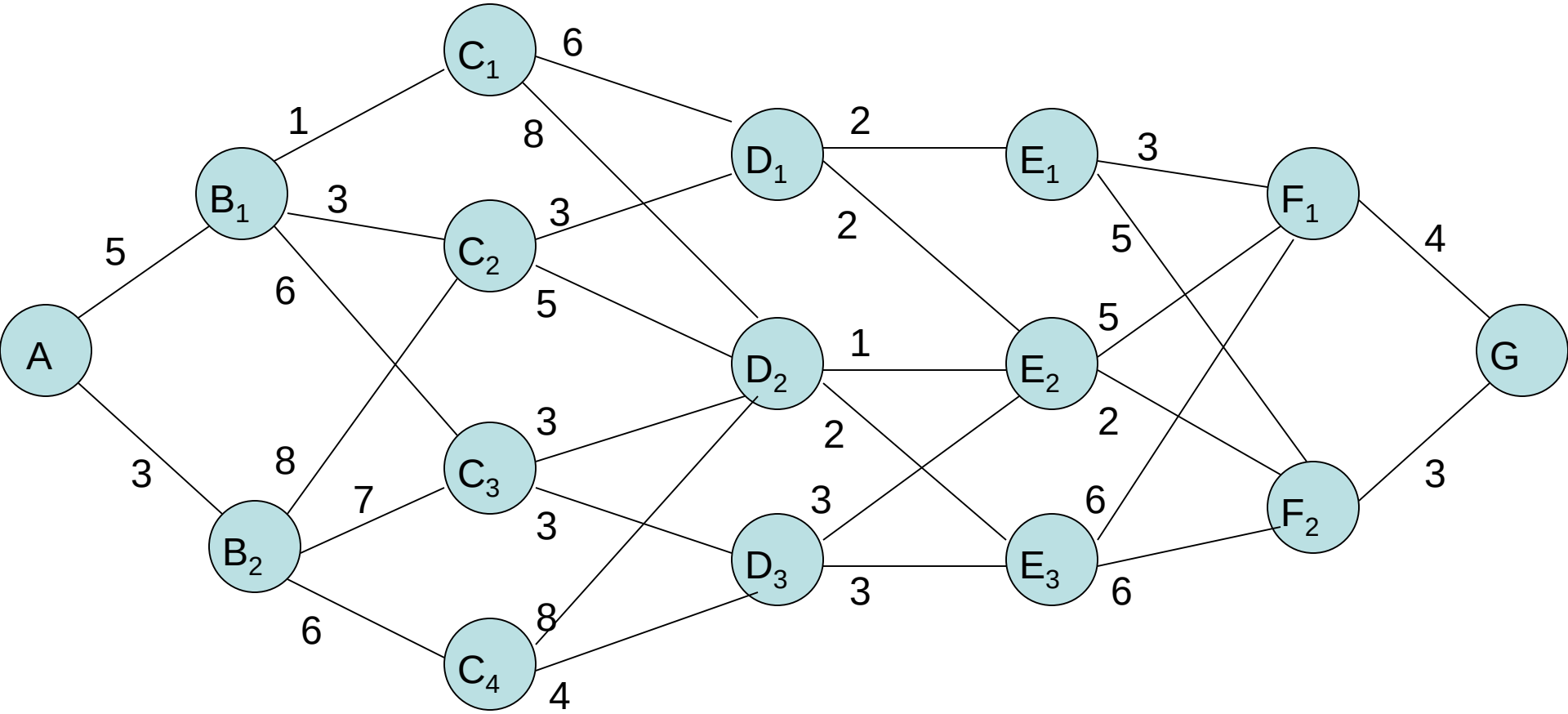
例1　最短路线问题

给出一个线路网络， 从A点要铺设一条管道到G点，其两点之间连线上的数字表示两点间的距离；

要求选择一条由A到G的铺管线路，使总距离为最短。

如图

最短路线问题


A
B1
B2
C1
C2
C3
C4
D1
D2
D3
E1
E2
E3
F1
F2
G
5
3
1
6
3
8
6
8
7
6
3
8
4
2
3
2
1
2
3
3
3
5
5
2
6
3
4
3
6

最短路线问题

- 从A到G可以分为6个阶段。各个阶段的决策不同，铺管路线就不同。
- 明显地，当某段的始点给定时，它直接影响着后面阶段的引进路线和整个路线的长短，而后面各阶段的路线的发展不受这点以前各段路线的影响。
- 问题的要求是：在各个阶段选取一个恰当的决策，使由这些决策组成的一个策略所决定的一条路线，其总路程最短----最优策略

穷举法：

共有2 * 3 * 2 * 2 * 2 * 1=48 条路线

动态规划的基本概念

阶段(Stage)

把所给问题的过程，恰当地划分成若干个相互联系的阶段，以便于求解。通常用k表示阶段变量。

例中第一阶段为A→B，包含2条支路：$A \rightarrow B_1$和$A \rightarrow B_2$

状态（State）

状态表示某段的出发位置。它既是该段某支路的始点，同时也是前一段某支路的终点。通常一个阶段包含若干个状态。

描述过程状态的变量，称为状态变量。常用x_k表示在第k段的某一状态。第k段状态集合可表示为

$$X_k = \{ x_k^{(1)}, x_k^{(2)}, \ldots, x_k^{(\mathrm{i})}, \ldots, x_k^{(\mathrm{r})}\}$$

故例中第三阶段的状态集合就可记为

$$X_3 = \{ x_3^{(1)}, x_3^{(2)}, x_3^{(3)}, x_3^{(4)}\} = \{1, 2, 3, 4\}$$

或 $$X_3 = \{C_1, C_2, C_3, C_4\}$$

动态规划的基本概念

决策（Decision）

- 决策就是某阶段状态给定以后，从该状态演变到下一阶段某状态的选择。
- 描述决策的变量，称为决策变量。
- 常用$u_k(x_k)$表示第k段当状态处于x_k时的决策变量。决策变量的取值往往限制在某一范围之内，此范围称为允许决策集合。通常以$D_k(x_k)$表示第k段当状态处于x_k时的允许决策集合。

$$u_k(x_k) \in D_k(x_k)$$

动态规划的基本概念

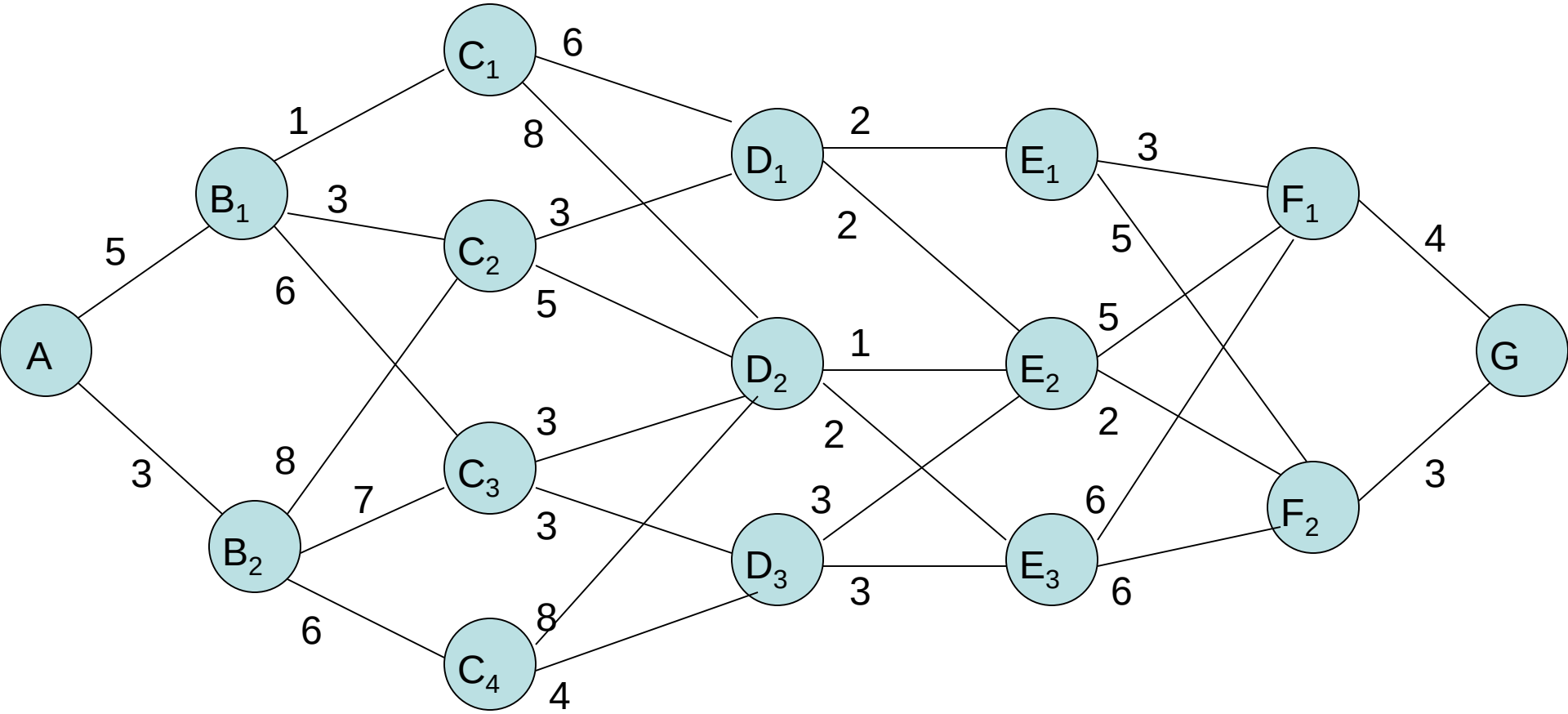
决策（Decision）

- 决策就是某阶段状态给定以后，从该状态演变到下一阶段某状态的选择。
- 描述决策的变量，称为决策变量。
- 常用$u_k(x_k)$表示第k段当状态处于x_k 时的决策变量。决策变量的取值往往限制在某一范围之内，此范围称为允许决策集合。通常以$D_k(x_k)$表示第k段当状态处于x_k 时的允许决策集合。

$$u_k(x_k) \in D_k(x_k)$$

例中第二阶段状态集合$X_2=\{B_1, B_2\}$；
则从B_1出发的允许决策集合$D_2(B_1)=\{C_1, C_2, C_3\}$；
若选择点C_2，则$u_2(B_1)=C_2$。

最短路线问题


A
B1
B2
C1
C2
C3
C4
D1
D2
D3
E1
E2
E3
F1
F2
G
5
3
1
6
3
8
6
8
7
6
3
8
4
2
3
2
1
2
3
3
3
5
5
2
6
3
4
3
6

动态规划的基本概念

策略（Policy）

- 由过程的第1阶段开始到终点为止的过程，称为问题的全过程。
- 由每段的决策$u_i(x_i)(i=1,2\ldots\ldots,n)$组成的决策函数序列就称为全过程策略，简称策略，记为$p_{1,n}$。

  即 $$p_{1,n}(x_k)=\{ u_1(x_1), u_2(x_2), \ldots\ldots, u_n(x_n)\}$$
- 由第k段开始到终点的过程称为原过程的后部子过程（或称为k子过程）。

  其决策函数序列$\{ u_k(x_k)，\ \ldots\ldots,\ u_n(x_n)\}$称为$k$子过程策略，简称子策略。即

  $$p_{k,n}(x_k)=\{u_k(x_k), u_{k+1}(x_{k+2}), \ldots\ldots, u_n(x_n)\}$$
- 在实际问题中，可供选择的策略有一定的范围，此范围称为允许策略集合，用P表示。从允许策略中找出的达到最优效果的策略称为最优策略。

动态规划的基本概念

指标函数

- 在多阶段决策过程最优化问题中，指标函数是用来衡量所实现过程的优劣的一种数量指标，它是一个定义在全过程和所有后部子过程上的确定数量函数，常用$V_{k,n}$表示。即

  $V_{k,n}= V_{k,n}(x_k ,u_k ,x_{k+1} ,\ldots,x_{n+1})\ \ k=1,2,\ldots,n$
- 不同的问题中，指标的含义也不同：距离、利润、成本、产量、资源消耗等。

  例中，$V_{k,n}$表示第k阶段由点x_k至终点G的距离；

  第k阶段由点x_k至$u_k(x_k)$的距离为**阶段指标**（阶段效益），可记为$d_k(x_k ,u_k)$。$d_5(E_1, F_1) = 3$。

动态规划的基本概念

最优指标函数

- 指标函数$V_{k,n}$的最优值，称为相应的最优指标函数。记为$f_k(x_k)$.

- 例中，$f_k(x_k)$表示从第k段x_k点到终点G的最短距离。

  如$f_4(D_1)$就表示从第4段中的D_1点到G 点的最短距离。

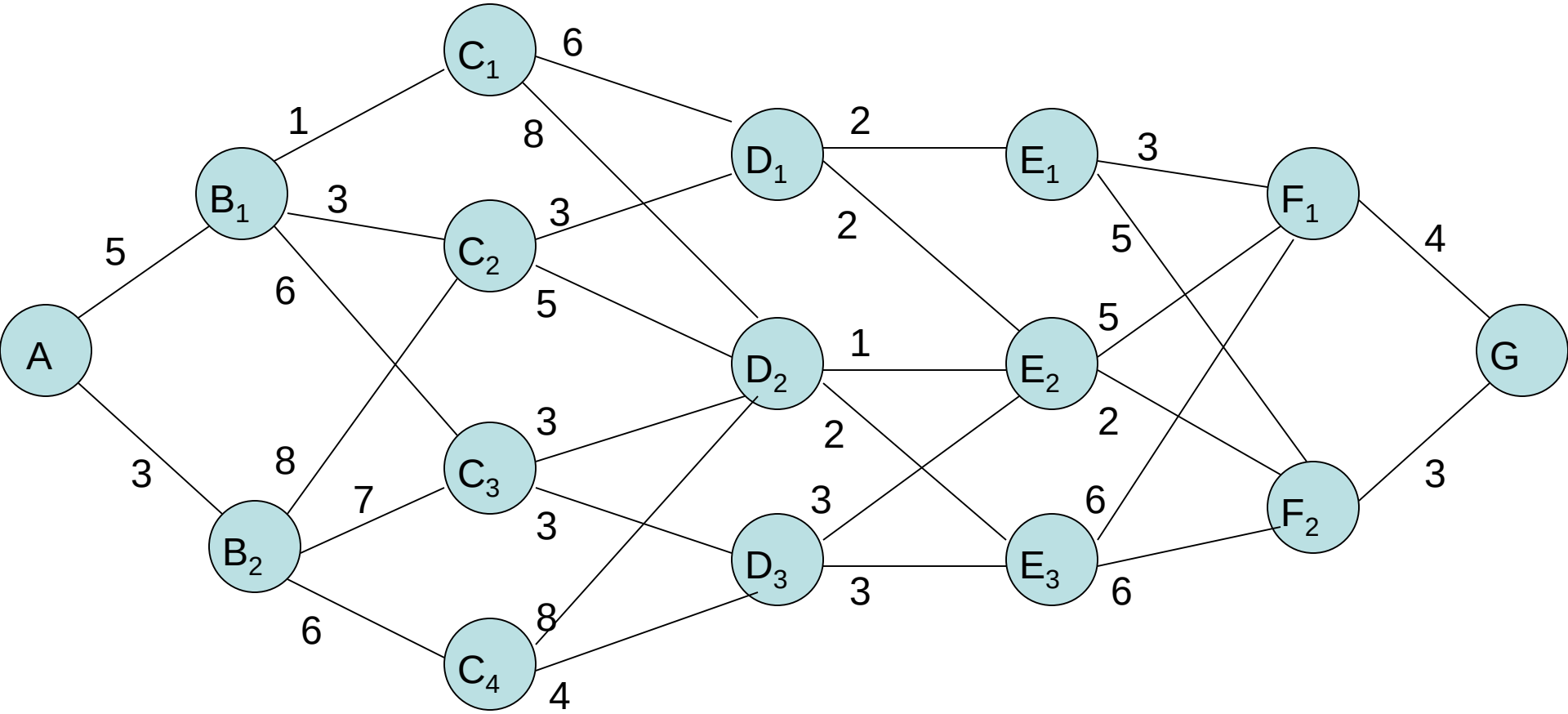
动态规划的基本思想和基本方程

- 生活中平常的事例，即可深刻揭示最短路线的重要特性：

  如果最短路线在第K站通过点P_k，则由点P_k出发到达终点的这条路线，对于从点P_k出发到达终点的所有可能选择的不同路线来说，必定也是最短路线。

- 动态规划的方法是从终点逐段向始点方向寻找最短路线的一种方法。

最短路线问题


A
B1
B2
C1
C2
C3
C4
D1
D2
D3
E1
E2
E3
F1
F2
G
5
3
1
6
3
8
6
8
7
6
3
8
4
2
3
2
1
2
3
3
3
5
5
2
6
3
4
3
6

最短路线问题

按照动态规划的方法，从最后一段开始计算，由后向前逐步推移至A点：

- 当$k = 6$ 时，$f_6(F_1)$表示在第6段由F_1至G的最短距离，故$f_6(F_1)=4$。同理，$f_6(F_2)=3$。
- 当$k = 5$ 时，若从E_1出发，则有两个选择，一是至F_1，一是至F_2，则

$$f_5(E_1)=\min\begin{Bmatrix} d_5(E_1,F_1)+f_6(F_1) \\ d_5(E_1,F_2)+f_6(F_2) \end{Bmatrix} = \min\begin{Bmatrix} 3+4 \\ 5+3 \end{Bmatrix} = 7$$

- 这说明，由E_1至终点G的最短距离为7，其最短路线是$E_1 \rightarrow F_1 \rightarrow G$，而相应的决策变量$u_5(E_1)=F_1$.

最短路线问题

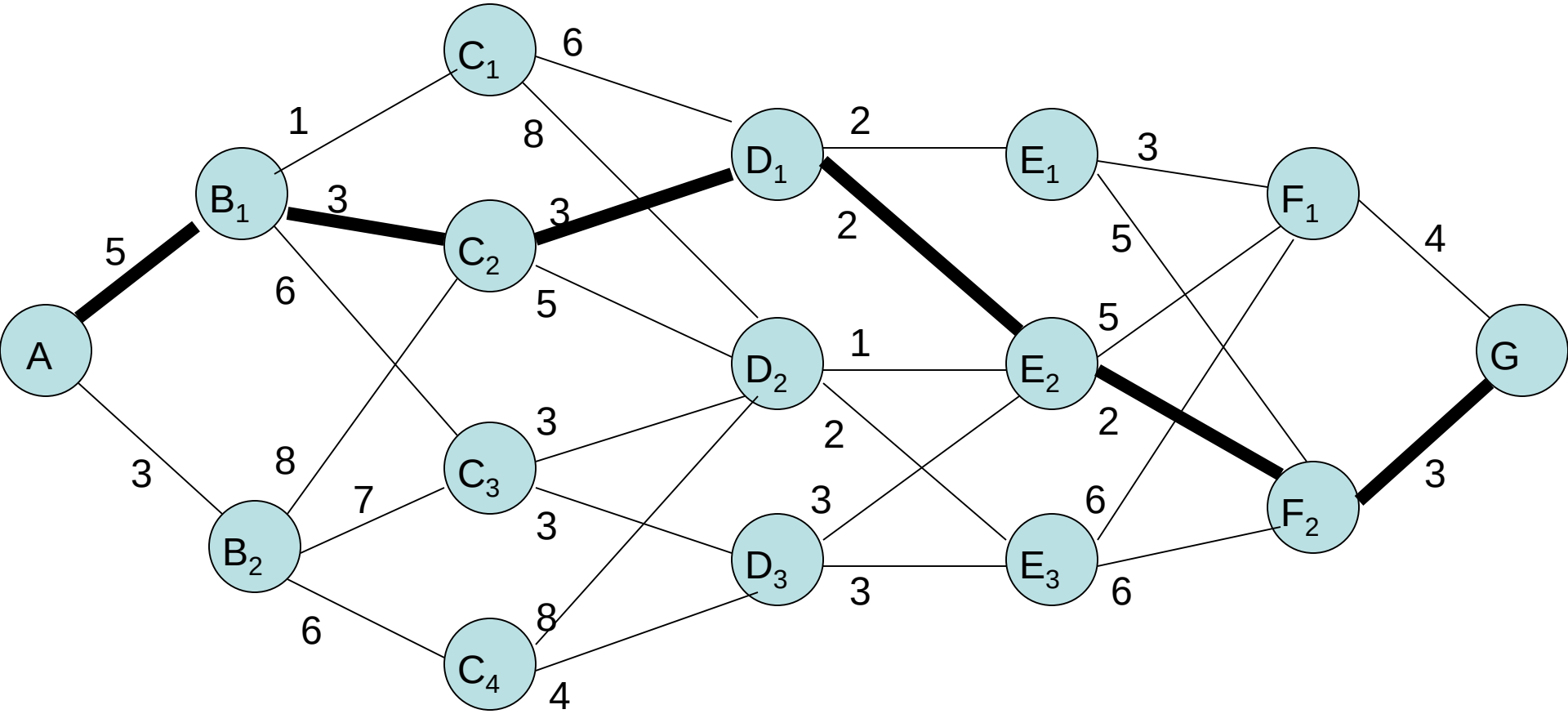
若从E_2出发，也有两个选择，一是至F_1，一是至F_2，则

$$f_5(E_2)=\min\begin{Bmatrix} d_5(E_2,F_1)+f_6(F_1) \\ d_5(E_2,F_2)+f_6(F_2) \end{Bmatrix} = \min\begin{Bmatrix} 5+4 \\ 2+3 \end{Bmatrix} = 5$$

- 这说明，由E_2至终点G的最短距离为5，其最短路线是$E_2 \rightarrow F_2 \rightarrow G$，而相应的决策变量$u_5(E_2)=F_2$.

- etc.

最短路线问题


A
B1
B2
C1
C2
C3
C4
D1
D2
D3
E1
E2
E3
F1
F2
G
5
1
6
8
3
3
6
5
3
8
7
3
3
6
8
4
2
2
1
2
3
3
3
5
5
2
6
3
4
6

动态规划的函数基本方程

- 在求解的各个阶段，我们利用了k阶段与k+1阶段之间的如下关系：

$$f_k(x_k) = \min_{u_k(x_k)}\{d_k(x_k,(u_k(x_k)) + f_{k+1}(u_k(x_k))\}$$

$$k = 6,5,4,3,2,1$$

$f_7(x_7)$= 0 (或写成 $f_6(x_6) = d_6(x_6, G)$)

动态规划最优化原理

- **动态规划最优化原理：**

  作为整个过程的最优策略具有这样的性质：即无论过去的状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略。

可逆过程

- 顺序解法：以G为始端，以A为终端的左行解法程序称为顺序解法
- 逆序解法
- 可以把段次颠倒过来求最优解的多阶段决策过程称为可逆过程。
- 顺序解法和逆序解法只表示行进方向的不同或始端的颠倒。但用动态规划方法求最优解时，都是在行进方向规定后，均要逆着这个规定的行进方向，从最后一段向前逆推计算，逐段找出最优途径。

与穷举法的对比

减少了计算量

- 穷举法：
  - 要对48条路线进行比较，比较运算要进行47次；
  - 求各条路线的距离（288次加法），即使使用逐段累加方法，也要进行0+6+12+32+48+48=146次加法运算。
- 动态规划方法：
  - 比较运算（从k=5段开始向前算）共进行3+3+4+4+1=15次。
  - 每次比较运算对应两次加法运算，再去掉中间重复两次（即$B1 \rightarrow C1$， $B2 \rightarrow C4$各多算了一次），实际只有28次加法运算。

与穷举法的对比

丰富了计算结果

在逆序解法中，使我们得到的不仅仅是由A点出发到终点G的最短路线及相应的最短距离；而且得到了从所有各中间点出发到终点的最短路线及相应的距离。

即，求出的不是一个最优策略，而是一族的最优策略。这对许多实际问题有重要意义。

构成动态规划模型的条件

建立动态规划模型时，除了要将实际问题恰当地划分若干个阶段（一般是根据时间和空间而划分的）外，主要明确以下四个方面：

1. 正确选择状态变量x_k，使它既能描述过程的状态，又要满足无后效性。

   动态规划中的状态与一般所说的状态概念是不同的，它必须具有三个特性：

- 要能够用来描述受控过程的演变特征。
- 要满足无后效性。 所谓无后效性是指：如果某段状态给定，则在这段以后过程的发展不受前面各阶段状态的影响。
- 可知性。即是规定的各段状态变量的值，由直接或间接都是可以知道的。

构成动态规划模型的条件

2 确定决策变量u_k及每段的允许决策集合$D_k(x_k)=\{u_k\}$。

3 写出状态转移方程

如果给定第k段状态变量x_k的值，则该段的决策变量u_k一经确定，第$k+1$段状态变量x_{k+1}的值也就完全确定。

x_{k+1}的值随x_k和u_k的值的变化而变化的这种对应关系，用

$$x_{k+1} = T_k(x_k, u_k)$$ 表示。

它表示由k段到$k+1$段的整体转移规律，称为状态转移方程。

构成动态规划模型的条件

4 根据题意，列出指标函数$V_{k,n}$关系，并要满足递推性。

正确列出指标函数关系，必须使它具有三个性质：

- 它是定义在全过程和所有后部子过程上的数量函数；
- 要满足递推关系。

即 $V_{k,,n}（x_k, u_k，x_{k+1}, ...，x_{n+1}）=$

$$\Psi[x_k, u_k, V_{k+1,n}(x_{k+1}, ..., x_{n+1})]$$

- $\Psi[x_k, u_k, V_{k+1,n}]$对其变元$V_{k+1,n}$来说要严格单调。

构成动态规划模型的条件

- 常见的指标函数是取各段指标和的形式。即

$$V_{k,n} = \sum_{j=k}^{n} v_j(x_j, u_j)$$

- 其中 $v_j(x_j, u_j)$ 表示第 j 段的指标。它显然是满足上述三个性质的。所以上式可写成

$$V_{k,n} = v_k(x_k, u_k) + V_{k+1,n}[x_{k+1}, \ldots, x_{n+1}]$$

构成动态规划模型的条件

- 当初始状态给定，过程的策略也确定了时，指标函数也就确定了。
- 因此，指标函数是初始状态和策略的函数。记为$V_{k,n}[x_k，p_{k,n}(x_k)]$。 故上面递推关系又可写成：

$$V_{k,n}[x_k, p_{k,n}] = v_k(x_k, u_k) + V_{k+1,n}[x_{k+1}, p_{k+1,n}]$$

构成动态规划模型的条件

- 因其子策略$p_{k,n}(x_k)$可看成是由决策$u_k(x_k)$和$p_{k+1,n}(x_{k+1})$组合而成。即

$$p_{k,n}(x_k)=\{u_k(x_k),\ p_{k+1,n}(x_{k+1})\}$$

- 对于最优策略的指标函数值，有

$$f_k(x_k)=V_{k,n}[x_k,\ p^*_{k,n}(x_k)]=\underset{p_{k,n}}{opt}V_{k,n}[x_k,\ p_{k,n}(x_k)]$$

- $P^*_{k,n}(x_k)$ 表示初始状态为x_k的后部子过程所有子策略中的最优子策略

动态规划的基本方程

- 由上述条件（组成部分）得出动态规划的基本方程：

- 若从x_k出发，有：

$$\underset{p_{k,n}}{opt}\, V_{k,n}(x_k, p_{k,n}) = \underset{\{u_k, p_{k+1,n}\}}{opt}\, \{v_k(x_k, u_k) + V_{k+1,n}(x_{k+1}, p_{k+1,n})\}$$

$$= \underset{u_k}{opt}\, \{v_k(x_k, u_k) + \underset{p_{k+1,n}}{opt}\, V_{k+1,n}\}$$

$$k = n, n\text{-}1, \ldots\ldots, 1$$

动态规划的基本方程

$$f_k(x_k) = \underset{p_{k,n}}{opt} V_{k,n}(x_k, p_{k,n})$$

$$= \underset{u_k}{opt} V_{k,n}(x_k, u_k, p^*_{k+1,n})$$

$$= \underset{u_k \in D_k(x_k)}{opt} \{v_k(x_k, u_k) + f_{k+1}(x_{k+1})\} \quad k = n, n-1...., 1$$

及 $f_{n+1}(x_{n+1}) = 0$

动态规划应用举例

资源分配问题

- 所谓资源分配问题，就是将供应量有限的一种或若干种资源（例如原材料、资金、机器设备、劳力、食品等等），分配给若干个使用者，而使目标函数为最优。
- 一维的分配问题

  设有某种原料，总数量为a， 用于生产n种产品。若分配数量x_i用于生产第i种产品，其收益为$g_i(x_i)$。问应如何分配，才能使生产n种产品的总收入最大？

$$\begin{cases} \max[g_1(x_1)+g_2(x_2)+.....+g_n(x_n)] \\ x_1+x_2+.....+x_n=a \\ x_i \geq 0 i=1,2.....,n \end{cases}$$

资源分配问题

- 通常以把资源分配给一个或几个使用者的过程作为一个阶段，把规划问题中的变量取为决策变量，将累计的量或随递推过程变化的量选为状态变量。
- 设状态变量x_k表示分配用于生产第k种产品至第n种产品的原料数量。
- 决策变量u_k表示分配给生产第k种产品的原料数，则，状态转移方程：

$$x_{k+1} = x_k - u_k$$

- 其中x_{k+1}表示分配用于生产第k+1种产品至第n种产品的原料数。
- 允许决策集合：$D_k(x_k) = \{u_k \mid 0 \le u_k \le x_k\}$
- $f_k(x_k)$表示以数量为x_k的原料分配给第k种产品至第n种产品所得到的最大总收入。
- 由最优化原理，有动态规划的递推关系式：

$$\begin{cases} f_k(x_k) = \max\limits_{0 \le u_k \le x_k} \{g_k(u_k) + f_{k+1}(x_k - u_k)\} k = n-1,.....,1 \\ f_n(x_n) = g_n(x_n) \end{cases}$$

资源分配问题—连续变量

- $g_i(x)$是线性函数、凸函数, $f_k(x)$易得。否则，
- 首先要把问题离散化。
- 把区间[0，a]进行分割，令x=0，△，2△，……, m△(=a)，其△的大小，应根据计算精度和计算容量来确定。
- 然后规定所有的$f_k(k)$只在这些分割点上取值，x_k也只取这些值。上式即为

$$\begin{cases} f_n(x_n) = g_n(x_n) \\ f_k(x_k) = \max\limits_{p=1,2,\ldots,q} [g_k(p\Delta) + f_{k+1}(x_k - p\Delta)] \qquad k = n-1,\ldots\ldots,1 \end{cases}$$

其中 x_k=q△。

生产与存储问题

- 一个生产项目，在生产批量、生产的成本费用、存储费用以及市场对产品的需求情况都是确定数值的条件下，为了根据实际需要制定计划，必须正确确定不同时期的生产量和库存量。

- 这是一个多阶段决策问题。我们把计划分为几个时期，视不同时期为不同的阶段。如果在某一阶段上，增大生产批量，则可以降低成本费，但因超过了该时期市场的需要量，就要存储一定数量的产品，因而增加了库存费用。

生产与存储问题

- 如果按市场不同时期市场的需要量来确定不同时期的产量，虽不必存储产品，不需要存储费用，但增加了每件产品的生产成本费用。

- 所以，不同时期的产量，就决定了该时期的生产成本和库存费用，同时又影响着下面那个时期的产量和费用。

- 因此，正确制定生产计划，确定各个时期的产量，使在几个时期内的生产成本和库存费用之和最小，这就是生产与存储问题的最优化目标，其约束条件就是在满足市场对该产品的需要量的同时，使库存量在整个计划末为零。

复合系统工作可靠性问题

- 若某种机器的工作系统由*N*个部件组成，只要有一个部件失灵，整个系统就不能正常工作。
- 这些部件的正常工作关系为串接关系，为提高系统工作的可靠性，在每一个部件上均装有主要元件的备用件，并且设计了备用元件自动投入装置。

- 显然，备用元件越多，整个系统正常工作的可靠性越大。但备用元件多了，整个系统的成本、重量、体积均相应加大，工作精度也降低。
- 因此，最优化问题是在考虑上述限制条件下，应如何选择各部件的备用元件数，使整个系统的工作可靠性最大？

复合系统工作可靠性问题

- 设部件i(i=1,2……,N)上装有z_i个备用元件时，它正常工作的概率为$p_i(z_i)$。则，整个系统正常工作的可靠性，可用它正常工作的概率衡量，即：

$$P=\prod_{i=1}^{N} p_i(z_i)$$

- 设装一个部件i的备用元件费用为c_i，重量为w_i，要求总费用不超过c，总重量不超过w，则这个问题的模型为：

目标函数 $$\max P(z_1,z_2,.....,z_n)=\prod_{i=1}^{N} p_i(z_i)$$

约束集合 $$R=\left\{z \mid \sum_{i=1}^{N} c_i z_i \le c, \sum_{i=1}^{N} w_i z_i \le w, z_i \ge 0\right\}$$

复合系统工作可靠性问题

- 这是一个整数规划问题，因z_i要求为整数；且目标函数是非线性的，非线性整数规划是个较为复杂的问题，但是用动态规划方法解这个问题是比较容易的。

- 为了构造动态规划模型，根据有两个约束条件，选二维状态变量，采用两个状态变量符号x_k, y_k来表达，其中

  x_k——由第k个到第N个部件所容许使用的总费用。

  y_k——由第k个到第N个部件所容许具有的总重量。

- 决策变量u_k为部件k上装的备用元件数z_k（即$u_k=z_k$）。（决策变量是一维的。）
- 状态转移方程为：
- $$x_{k+1} = x_k - u_k c_k$$
- $$y_{k+1} = y_k - u_k w_k \qquad (1 \le k \le N)$$

复合系统工作可靠性问题

- 允许决策集合为：

$$D_k(x_k, y_k) = \left\{ u_k \mid 0 \le u_k \le \min\left(\frac{x_k}{c_k}, \frac{y_k}{w_k} \right) \right\}$$

u_k为非负整数

- 指标函数为：

$$V_{k,n} = \prod_{i=k}^{N} p_i(u_i)$$

（$1 \le k \le N$）

- 由此可得整机可靠性的动态规划基本方程为：

$$\begin{cases} f_k(x_k, y_k) = \max\limits_{u_k \in D_k(x_k, y_k)} \{ p_k(u_k) f_{k+1}(x_k - u_k c_k, y_k - u_k w_k) \} \\ f_{N+1}(x_{N+1}, y_{N+1}) = 1 \end{cases}$$

- 边界条件为1，因为 x_{N+1}, y_{N+1} 均为零时，装置不工作，故可靠性为1。

复合系统工作可靠性问题

- 该问题的特点是：

  指标函数为连乘积形式，而不是连加形式，但仍满足递推关系；

  边界条件为1而不是零。它们是由研究对象的特性所决定的。

- 在该问题中，如果静态模型的约束条件增加为三个，例如总体积不超过v，则状态变量就要选为三维的（x_k,y_k,z_k）。它说明静态规划问题的约束条件增加时，对应的动态规划的状态变量维数也需要增加，而决策变量维数可以不变。