**贪心算法**

## 1、原理：在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的仅是在某种意义上的局部最优解。贪心算法不是对所有问题都能得到整体最优解，但对范围相当广泛的许多问题他能产生整体最优解或者是整体最优解的近似解。

## 2、特性：能够用贪心算法求解的问题一般具有两个重要特性：贪心选择性质和最优子结构性质。

**1)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。贪心算法则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。对于一个具体问题，要确定它是否具有贪心选择性质，必须证明每一步所作的贪心选择最终导致问题的整体最优解。

**2)最优子结构性质**

      当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。

## 3、贪心算法与动态规划算法的差异:

     动态规划和贪心算法都是一种递推算法，均有最优子结构性质，通过局部最优解来推导全局最优解。两者之间的区别在于：贪心算法中作出的每步贪心决策都无法改变，因为贪心策略是由上一步的最优解推导下一步的最优解，而上一部之前的最优解则不作保留，贪心算法每一步的最优解一定包含上一步的最优解。动态规划算法中全局最优解一定包含某个局部最优解，但不一定包含前一个局部最优解，因此需要记录之前的所有最优解。

4-1 会场安排问题

**理解问题**

要让使用的会场数最少，就需要每个会场的活动尽可能的多，即先让第一个会场安排的活动尽量多，再让第二个会场安排的活动尽量多，以此类推，直到所有活动被安排完毕。

**求解思路：**

用i代表第i个活动，s[i]代表第i个活动开始时间，f[i]代表第i个活动的结束时间，将活动按照结束时间进行从小到大排序。依次处理每一个活动，如果当前活动于所有已知的会场内活动都不相容，则增加一个新的会场，否则，遍历每个会场，看当前活动是否可以加入其中一个会场，是，则加入其中的**结束时间较晚**的会场，重复处理直到没有剩余会议。

**最优子结构性质：**

K是总的活动数量，设会场数目M是该问题的最优解，即需要安排{…}个会场，设中安排了p个活动，则剩余的（k-p）个活动所需会场的个数是M-1，如果对于(k-p)个活动存在更少的会场数量如m-2或更少，那个总共的k个活动所需要的会场数就是M-1，与M是最优解矛盾。

从另外一个角度看，将k个活动1,2,…,k看做坐标轴上的k个半开区间,i=1,2,…k。由于重叠的区间所对应的活动是不相容的，因此问题可以转换为求k个区间的最大重叠数。具体地，将k个活动的开始和结束时间共2k个值进行排序，然后从左到右扫描，遇到一个开始时刻，就将活动安排到一个空闲会场，或者增加一个会场，遇到一个结束时刻，就将活动从会场中释放，会场的状态变成空闲。该算法的主要时间是对2k个端点进行排序，时间复杂度是O(klogk)

**代码实现**

1. //活动安排问题 贪心算法
2. #include "stdafx.h"
3. #include <iostream>
4. using namespace std;
6. template<class Type>
7. void GreedySelector(int n, Type s[], Type f[], bool A[]);
9. const int N = 11;
11. int main()
12. {
13. //下标从1开始,存储活动开始时间
14. int s[] = {0,1,3,0,5,3,5,6,8,8,2,12};
16. //下标从1开始,存储活动结束时间
17. int f[] = {0,4,5,6,7,8,9,10,11,12,13,14};
19. bool A[N+1];
21. cout<<"各活动的开始时间,结束时间分别为："<<endl;
22. for(int i=1;i<=N;i++)
23. {
24. cout<<"["<<i<<"]:"<<"("<<s[i]<<","<<f[i]<<")"<<endl;
25. }
26. GreedySelector(N,s,f,A);
27. cout<<"最大相容活动子集为："<<endl;
28. for(int i=1;i<=N;i++)
29. {
30. if(A[i]){
31. cout<<"["<<i<<"]:"<<"("<<s[i]<<","<<f[i]<<")"<<endl;
32. }
33. }
35. return 0;
36. }
38. template<class Type>
39. void GreedySelector(int n, Type s[], Type f[], bool A[])
40. {
41. A[1]=true;
42. int j=1;//记录最近一次加入A中的活动
44. for (int i=2;i<=n;i++)//依次检查活动i是否与当前已选择的活动相容
45. {
46. if (s[i]>=f[j])
47. {
48. A[i]=true;
49. j=i;
50. }
51. else
52. {
53. A[i]=false;
54. }
55. }
56. }

**4-2 最优合并问题**

* **问题描述：**

给定k个排好序的序列,用2路合并算法将这k个序列合并成一个序列，假设所采用的2路合并算法合并2个长度为m和n的序列需要m+n-1次比较，试设计一个算法确定这个序列的最优合并顺序，使得所需要的总的比较次数最少。

* **算法思想**

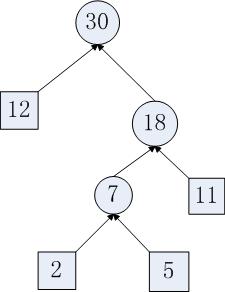
1. 最优合并顺序

将k个待合并序列按长度进行非减序排列， ，将前两个序列合并，并将合并后的新序列加入到剩下的序列中，并保持长度非减序排列，重复以上动作，直到所有序列合并完成。

1. 最差合并序列

按长度进行非增序排列，，将前两个序列合并，并将合并后的新序列加入到剩下的序列中，并保持长度非增序排列，重复以上动作，直到所有序列合并完成。

1. 问题转化



K个序列两两合并的过程可以用一颗完全二叉树T表示，如上图所示，在这颗二叉树中：

* 方形节点表示原始的序列，圆形节点表示合并过程中的序列
* 节点的权值表示序列的长度

由图不难看出，k个序列的合并总次数为 ，其中 为序列的长度，为在二叉树T中的深度，由于k-1为常数，欲使最小，只需使最小，于是该问题转化为最优前缀码问题，因为最优前缀码具有贪心选择性质和最优子结构性质，所以最优合并问题具有贪心选择性质和最优子结构性质。

* **代码实现**

#include <iostream>

#include <queue>

#include <string.h>

#include <vector>

using namespace std;

char Table[26];

struct Node

{

int freq;

char val;

Node \* left;

Node \* right;

Node():left(NULL), right(NULL) , freq(0) , val('0'){}

};

class Cmp

{

public:

bool operator() (const Node \* a, const Node \* b) const

{

return a->freq > b->freq; // 从小到大 ，freq 小的 优先级别高

}

};

priority\_queue<Node\* , vector<Node\*> , Cmp> myQueue;

void BuildTree()

{

for (int i = 0; i < 26; ++ i)

{

if (Table[i] > 0)

{

Node \* node = new Node();

node->freq = Table[i];

node->val =(char) (i + 'A');

myQueue.push(node);

}

}

while (myQueue.size() > 1)

{

Node \* f = myQueue.top();

myQueue.pop();

Node \* s = myQueue.top();

myQueue.pop();

Node \* tmp = new Node();

tmp->freq = f->freq + s->freq;

tmp->left = f;

tmp->right = s;

myQueue.push(tmp);

}

//cout << myQueue.top()->freq<<endl;

}

4-9汽车加油问题

问题描述：

一辆汽车加满油后可以行使nkm。旅途中有若干加油站，设计一个算法，指出应该在哪些加油站停靠，使得沿途加油次数最少。并证明算法能产生一个最优解。

算法设计思想：贪心算法，最远加油站优先

**贪心选择性质：**

设加油站编号0,1,2,…,k,k+1，其中0表示起点，k+1表示终点，加油站i到j的距离为,假设从0到k+1选取的加油站的最优解为()，记f为汽车从出发点加满油能达到的最远加油站编号，则有，当时，()是满足贪心选择性质的最优解，当时，由于从到的距离小于f到的距离，则汽车从f出发一定可以达到，所以()也是该问题的最优解，因此，总存在以贪心选择开始的最优加油方案。进一步，在做了贪心选择，即选择了加油站f之后，原问题就简化为从f开始的同样的子问题：在出发点加满油，在沿途加油站停靠使得加油次数最少，下面是该问题的最优子结构性质。

**最优子结构性质：**

设()是汽车加油问题的最优解，则易知，()是起点为，终点为k+1的汽车加油问题的最优解（加油次数为q-1）。如果不是，则说明存在次数少于q-1的解，这样，原问题就存在加油次数小于q的解，与q的最优性矛盾。因此，贪心选择策略获取的解一定是最优解。