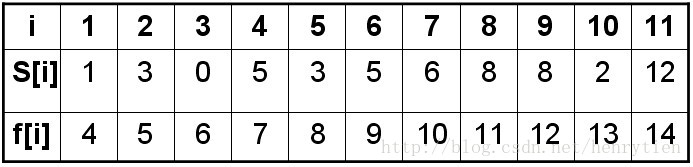
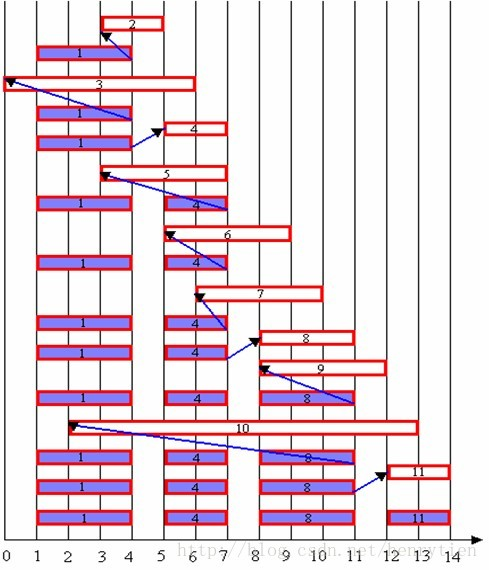


贪心策略：根据给的活动开始时间和结束时间，活动安排至少有三种看似合理的贪心策略可 供选择。  
  
(1)每次从剩下未安排的活动中选择具有最早开始时间且不会与已安排的活动重叠的活动来安排，这样可以增大资源的利用率。  
(2)每次从剩下未安排的活动中选择使用时间最短且不会与已安排的活动重叠的活动来安排。  
(3)每次从剩下未安排的活动中选择最早结束时间且不会与已安排的活动重叠活动来安排。  
通过比较上面三种策略，我们知道应该选用第三种，因为根据“活动结束时间=活动开始时间+使用资源时间”。

根据问题描述和所选用的贪心策略，对贪心策略求解活动安排的GreedySelector算法设计思路如下：  
(1)初始化。将k个活动的开始时间存储在数组S中；将k个活动的结束时间存储在数组F中且按照结束时间的非减序：f1≤f2≤…≤fn,数组S需要做相应的调整；采用集合A来存储问题的解，即所选择的活动集合，活动i如果在集合A中，当且仅当A[i]=ture.  
(2)根据贪心策略，算法GreedySelector首先选择活动1，即令A[1]=true.  
(3)依次扫描每一个活动，如果活动i的开始时间不小于最后一个选入集合A中的活动结束时间，即活动i和A中活动相同，则将活动i加入集合A中；否则，放弃活动i,继续检查下一个活动与集合A中活动相容性。



算法greedySelector 的计算过程如下图所示。图中每行相应于算法的一次迭代。阴影长条表示的活动是已选入集合A的活动，而空白长条表示的活动是当前正在检查相容性的活动。



若被检查的活动i的开始时间Si小于最近选择的活动j的结束时间fi，则不选择活动i，否则选择活动i加入集合A中。   
贪心算法并不总能求得问题的整体最优解。但对于活动安排问题，贪心算法greedySelector却总能求得的整体最优解，即它最终所确定的相容活动集合A的规模最大。这个结论可以用数学归纳法证明。

1. #include<stdio.h>
3. void greedy(int s[],int f[],int a[],int k);
5. int main()
6. {
8. int s[] = {1,3,0,5,3,5,6,8,8,2,12};
9. int f[] = {4,5,6,7,8,9,10,11,12,13,14};
10. int k;
11. k = sizeof(f)/sizeof(f[0]);
12. int \*a;
13. a = (int\*)malloc(sizeof(int)\*k);
15. greedy(s,f,a,k);
16. system("PAUSE");
18. }
20. /\*
21. \* s[]：活动的开始时间
22. \* f[]:活动的结束时间（非降序排列）
23. \* a[]:0或者1，为0表示活动不被安排，1表示活动被安排
24. \* k:活动个数
25. \*/
27. void greedy(int s[],int f[],int a[],int k)
28. {
29. int i;
30. int j = 0;
31. for(i=0;i<k;i++)
32. {
33. a[i] = 0;//初始所有活动都未被安排
34. }
35. a[0] = 1;
36. printf("第1个活动被安排\n");
37. int count = 1;
38. for(i=1;i<k;i++)
39. {
40. if(s[i] > f[j])
41. {
42. a[i] = 1;
43. printf("开始%d,结束%d.",s[i],f[i]);
44. j = i;
45. count++;
46. printf("第%d个活动被安排\n",i+1);
47. }
48. }
49. printf("总计%d个活动被安排\n",count);

52. }