一

(1)Θ "="

(2)O "<="

(3)o "<"

(4)Ω ">="

(5)ω ">"

二

b n=1

T(n)=

3T(n/2) + c\*n n>1

首先考虑n是2的整次幂，即对于某个正整数k，有n=2k。

T(n) = 3T(n/2) + c\*n

= 3(3T(n/4) + c\*n/2) + c\*n

= 32T(n/22) + 3 \*c\*n/2 + c\*n

= 33T(n/23) + (3/2)2 \*c\*n + 3/2 \*c\*n + c\*n

= …

= 3kT(1) +((3/2)k-1 + (3/2)k-2 + … + (3/2)2 + 3/2 + 1)c\*n

= b\*3k + 2\*c(3k-2k)

≤ (b+c)\*3k =О(3k)

因为，k=log2n，而xlogby = ylogbx

所以，T(n)= О(nlog23)。

a n=1，

T(n)=

2T(n/2) + c\*n n>1，

首先考虑n是2的整次幂，即对于某个正整数k，有n=2k。

T(n) = 2T(n/2) + c\*n

= 2(2T(n/4) + c\*n/2) + c\*n

= 4T(n/4) + 2c\*n

= 4(2T(n/8) + c\*n/4) + 2c\*n

= …

= 2kT(1) + k\*c\*n

= a\*n + c\*n\*log2n

所以，T(n)= О(nlogn)。

当n不是2的幂时，考虑到T(n)是n的不减函数，即T(2k)≤T(2k+1)，

所以，当2k≤n≤2k+1时，有T(n)≤T(2k+1)，所以仍有T(n)=О(nlogn)。

三

在折半查找过程的判定树上，定义根结点到每个内部结点(查找成功的结点)的路径长度之和为内部路径长度，记为I；定义根结点到每个外部结点(查找不成功的结点)的路径长度之和外部路径长度，记为E。 (15分)

试证明：1.具有n个内部结点的这样的判定树，满足n = ( E - I ) / 2 。

2.该判定树的高度为[log2n ] + 1，其中，[log2n ]表示取log2n 的整数。

答：1.设判定树的高度为h+1(含外部结点，且根结点的级别为1)。

根据描述折半查找过程的判定树的定义可知，判定树上不存在度为1的结点（即判定树是一个严格二叉树），且内部结点的最高层次为h；在h-1层以上是由内部结点构成的满二叉树。

不失一般性，假设第h层上有m个内部结点。则其余2h-1-m个结点是外部结点，且第h+1层上全部是外部结点。

则内部结点数n和m之间的关系为：n = 2h-1 - 1 + m ，即m = n - 2h-1 + 1

由于第i层的内部结点数为2i-1，根到第i层上的内部结点的路径程度为i-1(1≤i≤h-1)。

于是，根据外部路径长度和内部路径长度的定义可知：

I= ∑ (i-1)\*2i-1 + m\*(h-1) ，E = (2h-1-m)\*(h-1) + 2\*m\*h

所以，

E - I = (h-1)\*2h-1 -m\*(h-1) + 2\*m\*h - ∑ (i-1)\*2i-1 - m\*(h-1)

= (h-1)\*2h-1 -2m\*(h-1) + 2\*m\*h - ∑ (i-1)\*2i-1 = (h-1)\*2h-1 + 2m - ∑ (i-1)\*2i-1

= (h-1)\*2h-1 + 2(n - 2h-1 + 1) - ∑ (i-1)\*2i-1

令 S= ∑ (i-1)\*2i-1= 1\*2 + 2\*22 + 3\*23 +…+ (h-3)\*2h-3 +(h-2)\*2h-2

则，2S = 1\*22 + 2\*23 + 3\*24 +…+ (h-3)\*2h-2 +(h-2)\*2h-1

S - 2S = 2 + 22 + 23 + 24 +…+ 2h-2 - (h-2)\*2h-1

= 2\*(2h-2-1) - (h-2)\*2h-1

= 2h-1-2 - (h-2)\*2h-1

= - (h-3)\*2h-1 – 2

即，S = (h-3)\*2h-1 + 2

所以，有 E-I = (h-1)\*2h-1 + 2(n - 2h-1 + 1) - (h-3)\*2h-1 – 2 = 2n

即，n = ( E - I ) / 2 成立。

证毕

2．设具有n个结点的判定树的高度为h(含外部结点)。

对于任意的n＞1，则根判定树的构造规则可知，

判定树的第一层上有1个结点，即20=1个内部结点，

判定树的第二层上有2个结点，即21=1个内部结点，

一般地，若n≥2k，则第k层上有2k-1个内部结点。

显而易见，在高度为h的判定树的第h-2层上有2h-3个内部结点，在第h-1层上有2h-2个结点，其中部分是内部结点，部分是外部结点，且第h层的结点全部为内部结点。

于是，有

h-2

i=1

∑ 2i-1 = 2h-1-1 ＞ n

即有，2h-1≥ n

由于n＞1，h-1≥log2n

由于h是整数，所以有

h-1 = [log2n]，[log2n]表示取不大于log2n的整数值。

也即有h = [log2n] + 1。

证毕。

设有背包问题描述为：物品的重量为w1,w2,…,wn，对应的效益为pl,p2, …,pn，背包的容量为M。针对该背包问题，若p1/w1≥p2/w2≥…≥pn/wn并以此作为选择策略，试证明，依据贪婪算法Greedy\_Knapsack构造该问题的解，能够生成一个最优解。 (10分)

证明：设x=(x1, …, xn)是Greedy\_Knapsack所生成的解。如果所有的xi等于1，显然这个解就是最优解(效益值为Σpi)。

于是，设j是使xj≠1的最小下标。由算法可知，对于l≤i＜j，xi=1；对于j＜i≤n，xi=0；对于j，0≤xj＜1。

如果x不是一个最优解，则必定存在一个可行解y=(y1, …, yn)，使得Σpiyi＞Σpixi。

不失一般性，可以假定Σwiyi=M。且设k是使得yk≠xk的最小下标。显然，这样的k必定存在。

由上面的假设，可以推得yk＜xk。这个结论可从以下三个方面，即k＜j，k=j或k＞j分别加以证明。

① 若k＜j，则xk=1。因yk≠xk，从而yk＜xk；

② 若k=j，则由于Σwixi=M，且对于l≤i＜j，有xi=yi=1，而对于j＜i≤n，有 xi=0。

若yk＞xk，显然有Σwiyi＞M，与y是可行解矛盾；

若yk=xk，与假设yk≠xk矛盾，故yk＜xk。

③ 若k＞j，则Σwiyi＞M，这是不可能的。

现在，假定把yk增加到xk，那么必须从(yk+1, …, yn)中减去同样多的量，使得所用的总容量仍是M。这导致一个新解z=(z1, …, zn)，其中zi=xi，1≤i≤k，并且有

Σwi(yi-zi)=wk(zk-yk)。

因此，对于z有

Σpizi = Σpiyi + (zk-yk)wkpk/wk

-Σ(yi-zi)wipi/wi ≥ Σ piyi +

[(zk-yk)wk - Σ(yi-zi)wi]\*pi/wk

= Σpiyi

若Σpizi＞Σpiyi，则y不可能是最优解；若相等，且z=x，则x就是最优解；

若z≠x，则需重复上述讨论，或者证明y不是最优解，或者把y转换成x，从而证明了x也是最优解。

证毕。

证明，对于函数f(n)和g(n)，若*lim* g(n)/f(n)存在，则f(n)＝Ω(g(n))当且仅当存在确定的常数c，有*lim* g(n)/f(n)≤c。(10分)

证明：先证充分性：若f(n)=Ω(g(n))，则*lim* g(n)/f(n)≤c。 5分

根据Ω的定义，当f(n)=Ω(g(n))成立时，即存在正常数a和n0，使得对于所有的n≥n0，有f(n)≥a\*g(n)。

考虑到f(n)的非负性，以及a为正常数，所以有：

g(n)/f(n)≤1/a，

因为*lim* g(n)/f(n)存在，可令c=1/a，所以有因为*lim* g(n)/f(n)≤c。 充分性成立。

再证明必要性（有两种方法：一是根据定义推理；二是利用反证法）。

\因为*lim* g(n)/f(n)存在，可设为a，且有*lim* g(n)/f(n)≤c。

于是，根据极限定义有，对于给定的任意小的常数ε＞0，存在n1，使得对于一切n≥n1时，有

|g(n)/f(n)-a|≤ε，也即有-ε≤g(n)/f(n)-a≤ε，a-ε≤g(n)/f(n)≤ε+a，所以有

g(n)/f(n)≤ε+a，考虑到f(n)的非负性，所以有f(n)≥c\*g(n)，其中c=1/(ε+a)>0为常数。再取n0=n1,

也就是说，存在正常数c和n0，使得对于一切的n≥n0，有f(n)≥c\*g(n)，从而有f(n)=Ω(g(n))。

证毕。

试证明，当且仅当 *lim* f(n)/g(n)=0时，f(n)=o(g(n))。

证明：先证明若f(n)=o(g(n))，则*lim* f(n)/g(n)=0。 (5分)

根据小写o的定义，当f(n)=o(g(n))成立时，有f(n)=Ο(g(n))且f(n)≠Ω(g(n))。

即存在正常数c和n0，使得对于所有的n≥n0，有f(n)≤cg(n)。

考虑到f(n)的非负性，对于所有n≥n0，应有g(n)≥0。

若g(n)=0，则有f(n)≡0。根据大写Ο的定义及大写Ο的比率定理（定理2.2）可知，*lim* f(n)/g(n)存在且等于0。

n→∞

不失一般性，可以假设当n≥n0时有g(n)>0。根据大写Ο的比率定理（定理2.2）可知，f(n)/g(n)极限存在且*lim* f(n)/g(n)≤c。

n→∞

令*lim* f(n)/g(n) = a，其中0≤a≤c。若a=0，则问题得证。

假设a>0，则根据极限的有关知识，可导出矛盾。因为

*lim* g(n)/f(n) = *lim* (1/(f(n)/g(n)) = 1/a=c’。 c’是一个正的常数。

根据极限定义有，对于给定的任意小的常数ε>0，存在n1，使得对于一切n≥n1，

有| g(n)/f(n)-a|≤ε，也即有 g(n)/f(n)≤ε+a=d，从而有f(n)≥b\*g(n)，其中b=1/d，是一个常数。

于是，取m=max(n0,n1),则存在这样的正常数c，b，使得对于一切的n≥m，有

f(n)≤c\*g(n)和f(n)≥b\*g(n)都成立，这与小写o的定义矛盾。

所以，*lim* f(n)/g(n) = 0，

再证，若*lim* f(n)/g(n) = 0，则f(n)=o(g(n))。 (5分)

n→∞

根据极限定义，对于任意小给定的ε>0，存在n0，使得对于一切的n≥n0，有

| f(n)/g(n)| = f(n)/g(n)≤ε，即f(n)≤ε\*g(n)，取c=ε,满足大写Ο的定义。

即f(n)=Ο(g(n))

而*lim* g(n)/f(n) = *lim* 1/(f(n)/g(n))→∞,所以根据大写Ο比率定理，有

n→∞

n→∞

f(n)≠Ω(g(n))，即f(n)=o(g(n))。

证毕。

试证明：对于每一个无向连通图G，用Kruskal算法能够生成一棵最小成本的生成树。 (10分)

证明：设G是任一无向连通图，又设T是用Kruskal算法生成的G的生成树，再设T’是G的最小成本生成树。

现要证明T和T’具有相同的成本。

设E(T)和E(T’)分别是T和T’中的边集。

若n是G中的结点数，则T和T’都有n-1条边。

若E(T)=E(T’)，则T显然就是最小成本生成树，问题得证。

若E(T)≠E(T’)，则可假设e是一条使得e∈E(T)但不属于E(T’)的最小成本的边。显然，这样的e必定存在。

把e加到T’中后，必然产生一个唯一的环。

假设e, e1, e2, …, ek是这个唯一的环。这些边ei(1≤i≤k)至少有一条不在E(T)中，如若不然，则T也将包含环e, e1, e2, …, ek，这与T是树矛盾。

设ej是e, e1, e2, …, ek环中使得ej不属于E(T)的一条边，则有c(ej)＞c(e)，其中，c(e)是边e成本函数。

因为若ej比e有更小的成本，则Kruskal算法就会在e之前考虑ej，并把ej加到T中。

也就是说，E(T)中所有比e成本小的边都在E(T’)中，并且和ej一起构不成一个环。

因此，有c(ej)＞c(e)。

现在，重新考虑具有边集E(T’) ∪ {e}的图，去掉环e, e1, e2, …, ek上的任意一条边，则可得到一棵树T’’。

特别是，如果删去边ej，则所产生的树T’’的成本将不比T’的成本大，因为c(ej)≥c(e)。

因此，可得T’’也是一棵最小成本树。

若否，则反复使用上述的转换，树T’就可以转换成T而在成本上没有任何增加。

故T是一棵最小成本生成树。

证毕。

1. 已知f(n)=О(log2n)，求证：f(n)=О(logkn)，k＞1。 (6分)

证明：因为log2n= logkn / logk2

k＞1所以 logk2＞0

因为 f(n)＝О(log2n)，所以 存在正常数c和n0，使得对于一切n大于n0 有

f(n)≤c\*log2n＝(c/logk2) \* logkn

令c1＝c/logk2，则对于一切n大于n0 有

有f(n)≤c1\*log2n

所以，f(n)=О(logkn)

试证明Dijkstra算法的正确性。

证明：

Dijkstra算法用于求解不含负权值边的带权图G={V，E}中给定点*v*0到其余各个顶点的最短路径的算法，其中，V为顶点集，E为边集。

Dijkstra算法过程为：从给定源点*v*0出发，令S={*v*0}，计算*v*0顶点*v*i的最短路径，然后在所求得*n*-1条路径中，找一条最短路径*v*0, *w*，将*w*加入到S中。显然，由于顶点数n是有限的，这样的路径长度为1的最短路径是存在的，而且由于G中不存在负权值的边，这条路径在后面的计算中不会变得更短，即*v*0, *w*就是从*v*0到顶点*w*的最短路径。

考虑G的其余*n*-2个顶点的集合V-S，依据Dijkstra算法，对任意顶点*w*i∈V-S，重新计算从*v*0到*w*i的最短路径，这条路径必然是以下两条路径中的一条路径，即

或者是从*v*0到*w*i的边，或者是从*v*0经过S中的顶点到达*w*i的路径。

对于前者，无须证明；对于后者，很容易通过反证法得出这个结论。

再新计算的*n*-2条路径中，找一条最短的路径，并将对应的*w*i加入到S集合中。同样由于G中不存在负权值的边，所以从*v*0到*w*i的路径不会变的得更短。

针对集合V-S，重复上述过程*n*-3次，可以逐个顶点求出从*v*0到相应顶点的最短路径。因此，算法可以在有限步骤内终止，且每步的计算时间也是有限的，因为每躺最多计算*n*-1个顶点，一共进行*n*-1趟，故Dijkstra算法的时间复杂度为О(*n*2)。

特别地，当网络G中存在从*v*0出发不可达的顶点，则从*v*0到该顶点的最短路径为∞，它不会因为其余顶点的计算结果而发生改变。

综上，Dijkstra算法针对不含负权值边的带权图G={V，E}，能够在有限时间内计算出从源点到其余各个顶点的最短路径，切算法能够正常终止，所以，Dijkstra算法具有正确性。

针对含负权值边的图，Dijkstra算法不能保证求出从给定源点到其余各顶点的最短路径，这是保证Dijkstra算法正确性的前提。

证毕。

试证明：对于给定的不含负权值边的带权连通图，费洛伊德(Floyd)算法是正确的。

证明：

Floyd算法用于求解不含负权值边的带权图*G*={*V*，*E*}中任意两个顶点间的最短路径，其中，*V*为顶点集，*E*为边集。假设*V*中共有*n*个顶点。

针对给定的不含负权值边的带权连通图，Floyd算法依据动态规划的基本原理计算任意两点间的最短路径。

为叙述方便，用顶点的下标代表顶点。对任意的顶点对＜*i*, *j*＞，*i*≠*j*，计算从*i*到*j*的最短路径。定义顶点对＜*i*, *j*＞间的最短路径长度为0(当*i*=*j*时)，由于不存在负权值边，所以任意点顶对＜*i*, *j*＞的最短路径长度将大于等于0。

针对顶点对＜*i*, *j*＞，分别计算中间经过顶点的编号不大于1, 2, …, *n*的最短路径，并记录一条长度最短的路径，则该路径就是从顶点*i*到顶点*j*的最短路径。这是因为，在计算中间经过的顶点编号为*k*(1≤*k*≤*n*)的最短路径时，有

P*k*(＜*i*, *j*＞)=min{ P*k*-1(＜*i*, *j*＞), P*k*-1(＜*i*, *k*＞) + P*k*-1(＜*k*, *j*＞) }, 其中P*k*(＜*i*, *j*＞)表示从顶点*i*到顶点*j*，中间经过的顶点编号不大于*k*的最短路径。

由于满足最优性原理，所以P*k*(＜*i*, *j*＞)为P*k*-1(＜*i*, *j*＞)与P*k*-1(＜*i*, *k*＞) + P*k*-1(＜*k*, *j*＞)中的长度较小的值，其中，P*k*-1(＜*i*, *k*＞) 和 P*k*-1(＜*k*, *j*＞)分别表示从顶点*i*到顶点*k*与从顶点*k*到顶点*j*的中间经过顶点编号不大于*k*-1的最短路径。这样，只要计算出P*k*-1(＜*i*, *j*＞)、P*k*-1(＜*i*, *k*＞) + P*k*-1(＜*k*, *j*＞)以及P*k*(＜*i*, *j*＞)，就可计算出P*k*(＜*i*, *j*＞)。这个过程可从*k*=1起，依次计算到*n*，即可计算出从顶点*i*到顶点*j*的最短路径。

这个操作可在经过*n*次计算后得到终止，时间复杂度为О(*n*)。

因为要计算任意两顶点间的最短路径，所以只需要将上述过程嵌在对顶点*i*(1≤*j*≤*n*)与顶点*j*(1≤*j*≤*n*)的二重循环中即可完成，起时间复杂度为О(*n*3)。

即在*О*(*n*3)时间内，Floyd算法能够终止，并计算出任意两个顶点间的最短路径。因为*n*是有限的，所以Floyd算法能在有限时间内终止。

针对含负权值边的图，Floyd算法不能保证求出任意两个顶点之间的最短路径，所以，不含负权值的边是保证Floyd算法正确性的前提。

综上，Floyd算法针对不含负权值边的带权图*G*={*V*，*E*}，能够在有限时间内计算出从源点到其余各个顶点的最短路径，切算法能够正常终止，所以，Floyd算法具有正确性。

证毕。（红色字为要点部分，分别有分值2、4、4）

试证明：对于每一个带权连通无向图*G*，用*Prim*算法能够生成一棵最小成本的生成树。 (10分)

证明：设G是任一无向连通图，又设T是用*Prim*算法生成的G的生成树，再设T′是G的最小成本生成树。

现要证明T和T′具有相同的成本。

设E(T)和E(T′)分别是T和T′中的边集。

若*n*是G中的结点数，则T和T′都有*n*-1条边。

若E(T)=E(T′)，则T显然就是最小成本生成树，问题得证。

若E(T)≠E(T′)，则可假设*e*是一条使得*e*∈E(T)但不属于E(T′)的最小成本的边。显然，这样的*e*必定存在。

把*e*加到T′中后，必然产生一个唯一的环，设为*e*, *e*1, *e*2, …, *ek*。

在该环中，必然至少有一条*ei*(1≤*i*≤*k*)不在E(T)中。如若不然，则T也将包含环*e*, *e*1, *e*2, …, *ek*，从而与T是树矛盾。

设*ej*是*e*, *e*1, *e*2, …, *ek*环中使得*ej*不属于E(T)的那条边，则有*c*(*ej*) ≥*c*(*e*)，其中，*c*(*e*)是边*e*成本函数。

因为若*ej*比*e*有更小的成本，且选择*ej*或选择*e*都将构成图*G*的一棵生成树，那么，*Prim*算法就会在选择*e*之前将*ej*加入到T中。

也就是说，E(T)中所有比*e*成本小的边都在E(T′)中，并且和*ej*一起不构成环。

所以有c(*ej*) ≥c(*e*)。

现在，重新考虑具有边集E(T′)∪{*e*}的图，去掉环*e*, *e*1, *e*2, …, *ek*上的任意一条边，则可得到一棵树T′′。

显然，如果删去边*ej*，则所产生的树T′′的成本将不比T′的成本大，因为*c*(*ej*)≥*c*(*e*)。

因此，T′′也是一棵最小成本树。 8分

若T′′与T还存在不相同的边，则反复使用上述转换，就可以将树T′就转换成T而在成本上没有任何增加。 2分

故T是一棵最小成本生成树。

证毕。

|  |
| --- |
| bool bubble\_sort(int a[],int n) {  bool swapped = false;  for(int i = 0; i<n-1; i++) {  if(a[i]>a[i+1]) {  Swap(a[i],a[i+1]);  swapped = true;  };  return swapped;  }  void BubbleSort(int a[], int n) {  for(int i = n; i>1 && bubble\_sort( a, i ); i--);} |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s/e | 频率 | 总步数 |  |  |
| 0  0  1  1  1  3  1  0  1  0  0  0  1 | 0  0  θ (1)  θ (n)或n+1  θ (n)  Ω(0)，Ο(n)  Ω(0)，Ο(n)  0  θ (1)  0  0  0  Ω(n)，Ο(n2)  0 | 0  0  θ (1)  θ (n)  θ (n)  Ω(0)，Ο(n)  Ω(0)，Ο(n)  0  θ (1)  0  0  0  Ω(n)，Ο(n2)  0 |  |  |
|  | Ω(n),Ο(n2) |  |  |  |

2．最好、最坏时间复杂度分析 5分

最好情况是给定的待排序序列是正序的情形，此时不发生交换，只是内循环做了n次比较，故时间复杂度为Ο(n)。

最坏情况是给定的待排序序列是逆序的情形，此时外循环做n次，内循环还做n次比较，故时间复杂度为Ο(n2)。

|  |
| --- |
| MergeLinkList(LinkList &La , LinkList Lb) {  pa=La→next;pb= Lb→next;La→next = null;  LinkList s;  while(pa && pb) {  if(pa→data<=pb→data) {  s=pa;  pa=pa→next;}  else {  s=pb;  pb=pb→next; }  s→next = La→next ; La→next = s;  }//while;  if(!pa) pa=pb;  while(pa) {  s=pa; pa=pa→next;  s→next = La→next ; La→next = s;  }//while;  free(Lb);  }// MergeLinkList |

|  |  |  |
| --- | --- | --- |
| s/e | 频率 | 总步数 |
| 0  0  0  3  1  1  1  1  1  1  1  1  2  0  1  1  2  2  0  1  0 | 0  0  0  1  1  Ω(min(m,n))，О(m+n)  Ω(min(m,n))，О(m+n)  Ω(0)，О(m+n)  Ω(0)，О(m+n)  Ω(min(m,n))，О(m+n)  Ω(0)，О(m+n)  Ω(0)，О(m+n)  Ω(min(m,n))，О(m+n)  0  Θ(1)  Ω(1)，О(max(m,n))  Ω(2)，О(max(m,n))  Ω(2)，О(max(m,n))  0  1  0 | 0  0  0  3  1  Ω(min(m,n))，О(m+n)  Ω(min(m,n))，О(m+n)  Ω(0)，О(m+n)  Ω(0)，О(m+n)  Ω(min(m,n))，О(m+n)  Ω(0)，О(m+n)  Ω(0)，О(m+n)  Ω(min(m,n))，О(m+n)  0  Θ(1)  Ω(1)，О(max(m,n))  Ω(2)，О(max(m,n))  Ω(2)，О(max(m,n))  0  1  0 |
|  | О(m+n) |  |

|  |
| --- |
| Void SelectionSort(int a[],int n) {  bool sorted = false;  for(int size = n; !sorted && (size>1);size--) {  int pos = 0;  sorted = true;  for(int i=1;i<size;i++)  if(a[pos]<=a[i] pos = i;  else sorted = false; //未按序排列  swap(a[pos],a[size-1]);  }//for  }//SelectionSort |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| s/e | 频率 | 总步数 |  |  |  |
| 0  0  0  1  1  1  1  0  1  1  1  1  0  0 | 0  0  0  1  Ω(1),Ο(n)  Ω(0),Ο(n)  Ω(0),Ο(n)  0  Ω(n),Ο(n2)  Ω(n),Ο(n2)  Ω(n),Ο(n2) Ω(1),Ο(n)  0  0 | Θ(0)  Θ(0)  Θ(0)  Θ(1)  Ω(1),Ο(n)  Ω(0),Ο(n)  Ω(0),Ο(n)  Θ(0)  Ω(n),Ο(n2)  Ω(n),Ο(n2)  Ω(n),Ο(n2)  Ω(1),Ο(n)  Θ(0)  Θ(0) |  |  |  |
|  | Ω(n),Ο(n2) |  |  |  |  |

贪婪法：

**1，装箱问题：**

① j→remainder>=a[i] ② box\_volume- a[i] ③ box\_t=box\_t→next=j ；j→next = NULL

④ j→remainder-=a[i] ⑤ q→link=p

**2，马的遍历**

➀min=temp ➁kk=a[k] ➂j+=deltaj[no] ④board[i][j]=step

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 15 | 40 | 37 | 2 | 17 | 24 | 21 |
| 39 | 36 | 1 | 16 | 23 | 20 | 3 | 18 |
| 14 | 41 | 38 | 61 | 56 | 53 | 22 | 25 |
| 35 | 50 | 57 | 54 | 59 | 62 | 19 | 4 |
| 42 | 13 | 60 | 63 | 52 | 55 | 26 | 29 |
| 49 | 34 | 51 | 58 | 45 | 28 | 5 | 8 |
| 12 | 43 | 32 | 47 | 10 | 7 | 30 | 27 |
| 33 | 48 | 11 | 44 | 31 | 46 | 9 | 6 |

**3，TSP问题**

1. 由于针对每个顶点，在寻找其最小的下一个邻接顶点时，最多可能需要进行n次比较，即内循环最多要运行n次，所以，整个算法最多需要n\*n次计算，即时间复杂度为 О(n2) (3) 将每一个城市都作为出发点，在所有贪婪解中找最小的那个解，就是该TSP问题的最优解。 时间复杂度为О(n3)

**4, 硬币补充**

for(i=0；i<4；i++) {

b[i] = s/a[i]； //计算面值为a[i]的硬币枚数存于b[i]中 6分

s- = b[i]\*a[i]； //计算零钱余额 6分

}；

若 s=57，则找钱方案为： 3分

b[0]=2；b[1]=1；b[2]=1；b[3]=2 。

**5，零钱**

main()

{

int a[3]={25,10,5,1}； // a[0:3]存放硬币的不同面额

int b[4]； // b[0:3]存放对应硬币的数量(按面额由大到小)

int s； // 存放零钱总数

int i； // 循环控制变量

printf(“请输入零钱总数：”)；

scanf(“%d”,&s)；

for(i=0；i<4；i++) {

b[i] = s/a[i]； //计算面值为a[i]的硬币枚数存于b[i]中 5分

s-=b[i]\*a[i]； //计算零钱余额 5分

}；

for(i=0；i<4；i++) printf(“面额为%d的硬币需要%d枚 \n”，a[i]，b[i])；

}//main 3分

若 s=67，则找钱方案为 2分

b[0]=2；b[1]=1；b[2]=1；b[3]=2 。

**分治法：**

1，**循环比赛**

➀a[i][j]=a[i-twom1][j]+twom1 ➁a[i][twom1]=a[i-1][twom1]+1 ➂a[i][j]=a[i+1][j-1] ④a[a[i][j]][j]=i

2,字符串s反序

①：0 ②：len – (i + 1) ③：i<j ④：s[i] = s[j] ⑤：len

①：0 ②：len – 1 ③：i<j ④：s[i] = s[j] ⑤：--j

**哈密尔顿回路:**

①: x[k] = x[k] + 1;

②: c[x[k-1]][x[k]])==1 / 或 c[x[k-1]][x[k]]) ==1

③: k = k + 1;

④: k = k - 1;

⑤: s[x[k]] = 0;

**动态规划法**

**1，求最长子序列**

① c[0][j] = 0 ② c[i-1][j] ③ c[m][n] ④ c ⑤ j--

**回溯法：**

**1，求r个数组合**

① i==r-l ② a[i]=a[i-1]+1 ③ i==0 ④ i--;

① a[i]-i<=m-r+l ② i==r-1 ③ a[i]++ ④ i++ ⑤ a[--i]++

**2，八皇后**

① m == n；② a[col[m] ] = b[m+col[m] ] = c[n+m-col[m] ] = 1③ b[ m+col[m] ]

④ c[ n+m-col[m] ]