背包

```
/*
本关任务：给定N个物品和一个背包，背包的容量为W， 假设背包容量范围在[0，15]，第i个物品对应的体积和价值分别为W[i]和v[i]。各种物品的价值和重量如下：
     物品编号   1   2   3   4   5
      重量W    3   4   7   8   9
      价值V    4   5   10  11  13
求: 如何选择装入背包的物品，使得装入背包的物品的总价值为最大。

*/
#include <stdio.h>

int content[6]={0};              //最优解的物品组成
int w[6]={0,3,4,7,8,9};          //物品对应的重量
int v[6]={0,4,5,10,11,13};       //物品对应的价值
int bV=15;                       //背包的最大容量为15
int maxVal[6][16]={0};          //存放当物品数为i,背包容量为j的最大总价值

void findContent(int i, int j);  //找到最优解的物品组成

void findMax();                 //寻找当物品数为i,背包容量为j时的最大总价值

int main( void )
{
    int i,j;
    printf("w v 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15\n");
    findMax();
    for (int i=0;i<6;i++)  //打印当物品数为i,背包容量为j时的最大总价值
    {
        printf("%d %d ",w[i],v[i]);
        for(int j=0;j<16;j++)
        {
            printf("%d",maxVal[i][j]);
            if(j!=15){
                    printf(" ");
            }
        }
        printf("\n");
    }

    return  0;
}

void findMax() //寻找当物品数为i,背包容量为j时的最大总价值
{
    for(int i=1;i<6;i++)
        for(int j=1;j<16;j++)
 {
       if (j < w[i])       //如果背包容量小于物品i重量，表示背包存放不下第i种物品,此时的最大总价值为i-1种物品的最大总价值
                maxVal[i][j] = maxVal[i - 1][j];
        else
    {
        if(maxVal[i-1][j]>(maxVal[i-1][j-w[i]]+v[i]))//放下第i种物品时的总价值为第i种物品的价值加上当物品数为i-1背包容量为j-w[i]的最优解
```

```
            maxVal[i][j]=maxVal[i-1][j];             // 对比当放下第i种物品时的总价值和物品数位i-1
时的总价值，取最大值
        else
            maxVal[i][j]=maxVal[i-1][j-w[i]]+v[i];
    }

 }

}
```

DFS 八皇后

```
#include<stdio.h>
#include <stdlib.h>
#include <stdio.h>
#define N 8
int queen[N] = {0}; //每行的第几个位置
int count = 0;
int check(int row, int col) {
    for (int i = 0; i < row; i++) {
        if (queen[i] == col || row - i == col - queen[i] || row - i == queen[i] - col) {
            return 0;
        }
    }
    return 1;
}
void backtrack(int row) {
    if (row == N) {
        count++;
        return;
    }
    for (int i = 0; i < N; i++) {
        if (check(row, i)) {
            queen[row] = i;
            backtrack(row + 1);
            queen[row] = 0;
        }
    }
}
//用回溯法编程实现八皇后问题求解
int main()
{
/************ Begin ************/
    backtrack(0);
    printf("%d", count);
/************ End *************/
return 0;
}
```

快排+二分

```
/*
本关任务：随机生成20个从1－100之间的随机数，用递归与分治法编程实现元素的查找算法。
*/
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define N 10

void merge_sort_recursive(int arr[], int reg[], int start, int end) {
    if (start >= end)
        return;
    int len = end - start, mid = (len/2) + start;
    int start1 = start, end1 = mid;
    int start2 = mid + 1, end2 = end;

    merge_sort_recursive(arr, reg, start1, end1);
    merge_sort_recursive(arr, reg, start2, end2);

    int k = start;
    while (start1 <= end1 && start2 <= end2)
        reg[k++] = arr[start1] < arr[start2] ? arr[start1++] : arr[start2++];
    while (start1 <= end1)
        reg[k++] = arr[start1++];
    while (start2 <= end2)
        reg[k++] = arr[start2++];
    for (k = start; k <= end; k++)
        arr[k] = reg[k];
}

void merge_sort(int arr[], const int len) {
    int reg[len];
    merge_sort_recursive(arr, reg, 0, len - 1);
}

void search(int a[],int target,int p,int r)
{
    int mid =(p+r)/2;
    if(target > a[mid])
    {
        p=mid+1;
    }
    if(target < a[mid])
    {
        r=mid-1;
    }
    if(target==a[mid])
    {
        printf("%d",a[mid]);
        return;
    }
    else
    {
       search(a,target,p,r);
    }
}
void searchD(int a[],int target,int p,int r){
    if(p>=r){
        return -1;
    }
    int mid=(p+(r-p))/2;
    if(target==a[mid]){
        return mid;
    }
    if(target>a[mid]){
        searchD(a,target,p,mid-1);
    }else{
        searchD(a,target,mid,r);
    }
}
int main( void )
{
    int a[20];
    for(int i=0;i<10;i++){
        scanf("%d",&a[i]);
    }
    int target;
    scanf("%d",&target);
    merge_sort(a,10);
    for(int j=0; j<N; ++j){
        printf("%d",a[j]);
        if(j < N-1) {
            printf(" ");
        }
    }
    printf("\n");
    search(a, target, 0, 9);
    return 0;
}
```

贪心 最小延迟调度

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100

// 结构体定义：客户请求
struct Request {
    int time;       // 服务时间
    int ddl;        // 期望完成时间
    int begin;      // 实际开始时间
    int end;        // 实际完成时间
};

// 比较函数，按照期望完成时间排序
int compare(const void *a, const void *b) {
    const struct Request *req1 = (const struct Request *)a;
    const struct Request *req2 = (const struct Request *)b;
    return req1->ddl - req2->ddl;
}

// 快速排序函数
void quicksort(struct Request req[], int low, int high) {
    if (low < high) {
        int pivot = low;
        int i = low;
        for (int j = low + 1; j <= high; ++j) {
            if (req[j].ddl < req[pivot].ddl) {
                i++;
                // Swap req[i] and req[j]
                struct Request temp = req[i];
                req[i] = req[j];
                req[j] = temp;
            }
        }
        // Swap req[i] and req[pivot]
        struct Request temp = req[i];
        req[i] = req[pivot];
        req[pivot] = temp;

        // Recursively sort the two halves
        quicksort(req, low, i - 1);
        quicksort(req, i + 1, high);
    }
}

// 计算最小延迟调度的最大延迟时间，并输出服务顺序
int minimum_delay_schedule(struct Request req[], int requestNum) {
    quicksort(req, 0, requestNum - 1);

    int maxDelay = 0;
    int current_time = 0;

    for (int i = 0; i < requestNum; ++i) {
        int service_time = req[i].time;
        int ddl = req[i].ddl;

        int start_time = (current_time <= ddl - service_time) ? ddl - service_time :
current_time;

        int completion_time = start_time + service_time;

        int delay = (completion_time > ddl) ? completion_time - ddl : 0;
        if (delay > maxDelay) {
            maxDelay = delay;
        }

        current_time = completion_time;

        req[i].begin = start_time;
        req[i].end = completion_time;

        printf("%d ", i + 1);
    }

    printf("\n");

    return maxDelay;
}

// 主函数，处理输入和输出
int main() {
    struct Request req[MAX_SIZE]; // 客户请求数组
    int requestNum; // 请求数量

    printf("输入请求数量: ");
    scanf("%d", &requestNum);

    if (requestNum > MAX_SIZE) {
        printf("请求数量过多\n");
        return 0;
    }

    // 输入每个请求的时间和期望完成时间
    for (int i = 0; i < requestNum; ++i) {
        printf("请求 %d 的时间和期望完成时间: ", i + 1);
        scanf("%d %d", &req[i].time, &req[i].ddl);
    }

    // 调用最小延迟调度函数
    int min_max_delay = minimum_delay_schedule(req, requestNum);

    // 输出服务顺序
    printf("服务顺序: ");
    for (int i = 0; i < requestNum; ++i) {
        printf("%d ", i + 1);
    }
    printf("\n");

    // 输出最小的最大延迟时间
    printf("最小的最大延迟: %d\n", min_max_delay);

    return 0;
}
```