演算法作業 4

- Greedy 貪婪演算法

Pseudo code:

```
int main()
{
    int i,j,n,num,num1;
    int temp,temp1,temp2;
    cin>>n;
    int a[3][n];
    for(i=0;i<n;i++){
        cin>>a[i][0];
        cin>>a[i][1];
        a[i][2]=i+1;
    }
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
          if(a[i][0]<a[j][0] || a[i][1]<a[j][1] || a[i][2]<a[j][2]){
            swap
            swap
            swap}
    }
    int time;
    std::vector<int>dp;
    int profit[n];

    dp.push_back(a[0][2]);
    for(j=0;j<n;j++){
        int counting=0;
        for(i=j;i<n;i++){
            time=a[i][0];
            if(time+a[i+1][0]<=a[i+1][1]){
                time=time+a[i+1][0];
                dp.push_back(a[i+1][2]);
                counting=counting+1;}
            }
        profit[j]=counting;
    }

    for(i=0;i<dp.size()-1;i++){
        cout<<profit[i]<<" ";
    }
    cout<<endl;

    for(i=0;i<dp.size()-1;i++){
        cout<<dp[i]<<" ";
    }
    cout<<dp[dp.size()-1];
    return 0;
}
```

我是先把輸入資料轉換為二維陣列，然後根據所花費時間

(proceeding time)去排序，完成後花費最少時間是在第一列

雙層迴圈依序加入每一個去比較是否有小於規定截止時間

如果有，則是將計數器+1，並設定當前時間為 current，最後將計數器加入某一陣列，比較陣列中最大效益(profit)這樣便可以找出是哪一個擁有最大效益的工作排程，最後將 current 依序扣減 proceeding time 最後直到 0，將對應編號輸出

- DP 動態規劃演算法

pseudo code:

```
# 演算法作業4-DP
 /pseudo code/


int knapSack(int W, int wt[], int val[], int n)
{
    int i,w;
    vector<vector<int>> K(n + 1, vector<int>(W + 1));
    int dp[n+1][w+1];

    // Build table K[][] in bottom up manner
    for(i = 0; i <= n; i++)
    {
        for(w = 0; w <= W; w++)
        {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] +
                              K[i - 1][w - wt[i - 1]],
                              K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }
    return K[n][W];
    final(K[n][W],wt[],4);
}

void final(int K[n][W],int wt[],int n){
          sort(wt[]);
          currenttime = wt[0];
          for(int i=0;i<n;i++){
            currenttime = currenttime+wt[i];}
          if(currenttime=K[n][W]){
           for(int i=0;i<K[n][W]){cout<<wt[i];}}
          else(){
                  }
}
```

```
int main()
{
    int i,j,n;
    cin>>n;
    int val[n];
    int a[n][2];
    int wt[n];
    int maximum[n];
    for(i=0;i<n;i++){
        val[i]=1;
    }
    for(i=0;i<n;i++){
        for(j=0;j<2;j++){
            cin>>a[i][j];
        }
    }
    for(i=0;i<n;i++){
        maximum[i]=a[i][1];
    }
    for(i=0;i<n;i++){
        wt[i]=a[i][0];
    }
    int W = *max_element(maximum , maximum+n);
    cout << knapSack(W, wt, val, n);
    return 0;
}
```

想法:這題 DP 跟 0-1 knapstack 的操作很有相似之處，我採用把原本用的陣列表格改用 W:最大允許結束時間,wt[]存取所有工作的所需時間,val[]則是因為這題每項工作權重都是 1,沒有大小問題，所以是

[1,1,1…….1],然後是最後 n=4 根據使用者輸入去看 n=4 有 4 筆資料

最後我沒有把工作排序做出來，原本想說建立函式

```
void final(int K[n][W],int wt[],int n){
            sort(wt[]);
            currenttime = wt[0];
            for(int i=0;i<n;i++){
              currenttime = currenttime+wt[i];}
            if(currenttime=K[n][W]){
             for(int i=0;i<K[n][W]){cout<<wt[i];}}
            else(){
                      }
}
```

去把 current time 設定為依序增加的工作時間，依序扣減找出對應編號，但目前還沒想到要怎麼做