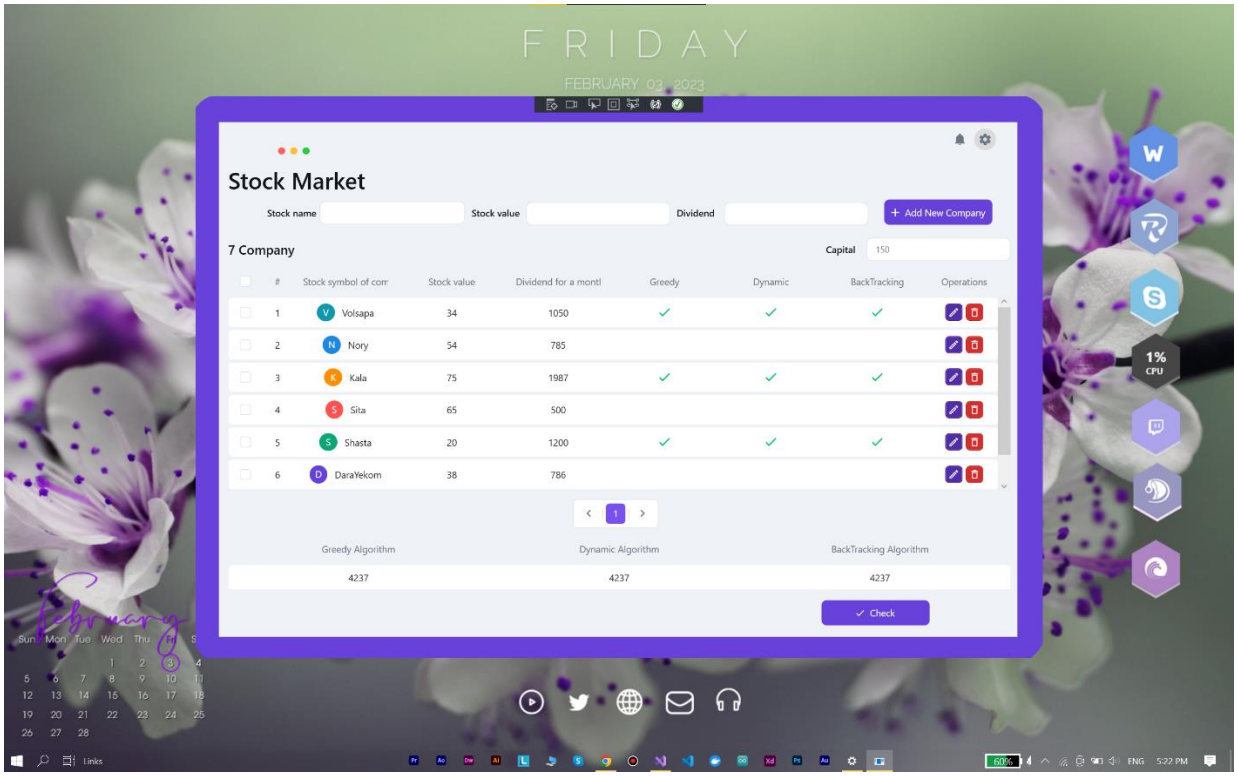
به نام خدا

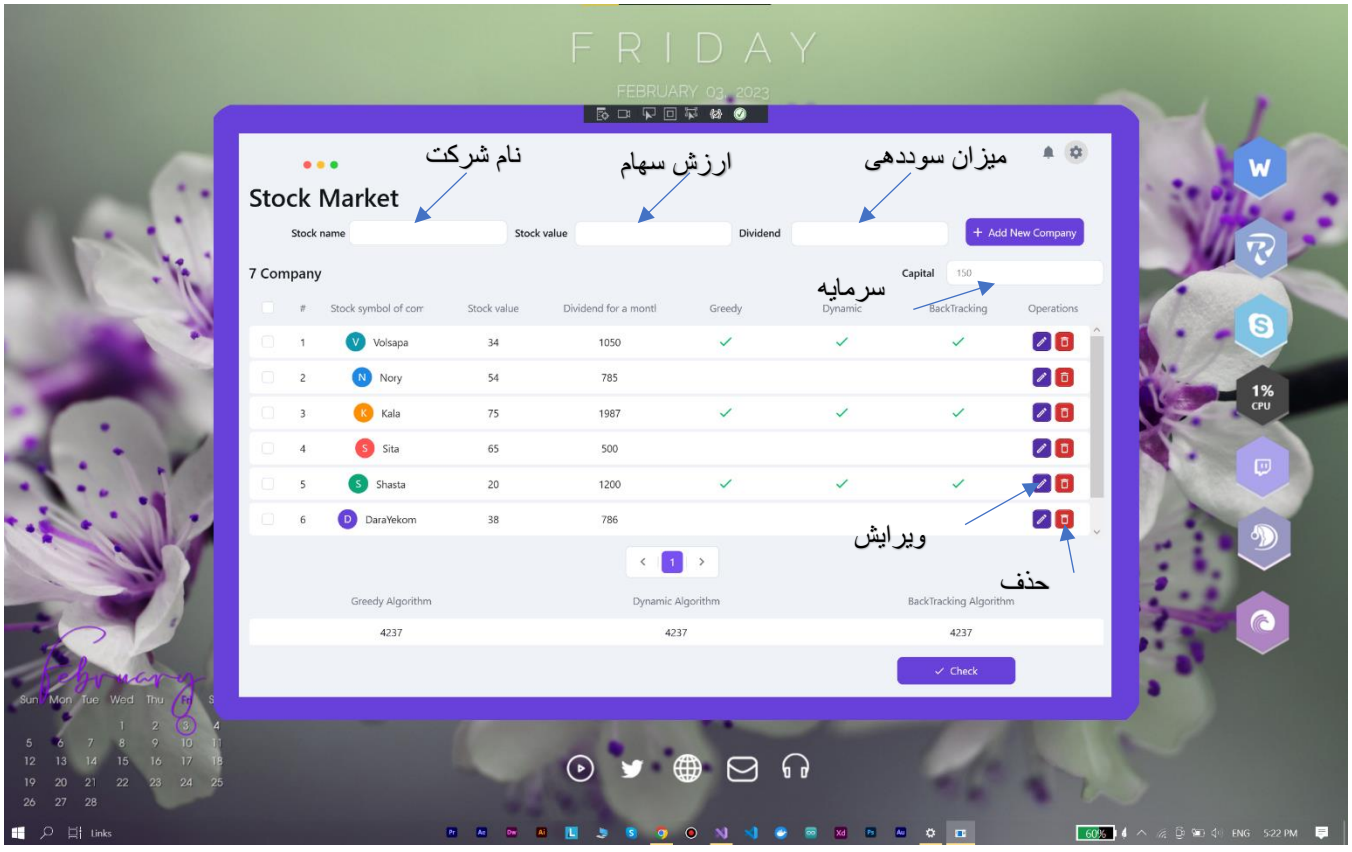
پروژه : کوله پشتی 0و1(بورس)

نام و نام خانوادگی : علی بابارهایی

شماره دانشجویی : 992164022

دموی برنامه


FRIDAY
FEBRUARY 03, 2023
Stock Market
Stock name
Stock value
Dividend
+ Add New Company
7 Company
Capital
150
Stock symbol of corr Stock value Dividend for a montl Greedy Dynamic BackTracking Operations
1 Volsapa 34 1050
2 Nory 54 785
3 Kala 75 1987
4 Sita 65 500
5 Shasta 20 1200
6 DaraYekom 38 786
Greedy Algorithm
Dynamic Algorithm
BackTracking Algorithm
4237
4237
4237
✓ Check


FRIDAY
FEBRUARY 03, 2023
نام شرکت
ارزش سهام
میزان سوددهی
Stock Market
Stock name
Stock value
Dividend
+ Add New Company
7 Company
Capital
150
سرمایه
Stock symbol of corr Stock value Dividend for a montl Greedy Dynamic BackTracking Operations
1 Volsapa 34 1050
2 Nory 54 785
3 Kala 75 1987
4 Sita 65 500
5 Shasta 20 1200
6 DaraYekom 38 786
ویرایش
حذف
Greedy Algorithm
Dynamic Algorithm
BackTracking Algorithm
4237
4237
4237
✓ Check

الگوریتم Greedy

این الگوریتم بر اساس حاصل تقسیم ارزش(Value) بر وزن (Weight) انتخاب میکند براین اساس که هر چی حاصل این تقسیم بیشتر باشد اون شی را انتخاب میکند

این الگوریتم همیشه جواب بهینه نمی دهد

```
public class Greedy_Algorithm
    {

        public double totalVal = 0d;
        public List<long> company_list=new List<long>();
        public double KnapSack_Greedy(item[] items, long w)
        {


            cprCompare cmc = new cprCompare();
            Array.Sort(items, cmc);                  //Time Complexity O(nlogn)


            long currentW = 0;
            foreach (item i in items)                //Time Complexity O(n)
            {
                long remaining = w - currentW;
                if (i.weight <= remaining)
                {
                    totalVal +=i.value;
                    currentW += i.weight;
                    company_list.Add(i.id);
                }
                if (remaining == 0)
                    break;
            }
            return totalVal;
        }
    }
```

Time Complexity : $O(n\log n)+O(n)=O(n)$

Space Complexity : $O(n)$

الگوریتم Dynamic

روش 1

در این الگوریتم از پایین به بالا حرکت میکنیم (برعکس Divide & Conquer)
برای پیاده سازی از آرایه دو بعدی استفاده میکنیم

```
public class Dynamic_Algorithm
   {
       public double totalVal = 0;
       public List<long> company_list = new List<long>();
       public void R_Companies(long n, long total_W, long[,] k, item[] items)
       {
           long current_value = 0;
           long preview_value = 0;
           for (long j = n; j > 0; j--)            //Time Complexity  O(n)
           {
               current_value = k[j, total_W];
               preview_value = k[j - 1, total_W];
               if (current_value > preview_value)
               {
                   company_list.Add((items[j - 1].id));
                   total_W = total_W - items[j - 1].weight;
               }
           }
       }
       public void knapSack_Dynamic(item[] items, long W)
       {
           long n = items.Length;
           long[,] K = new long[n + 1, W + 1];
           for (long i = 0; i <= n; i++)            //Time Complexity  O(N*W)
           {
               for (long w = 0; w <= W; w++)
               {
                   if (i == 0 || w == 0)
                       K[i, w] = 0;
                   else if (items[i - 1].weight <= w)
                       K[i, w] = Math.Max(items[i - 1].value + K[i - 1, w -
                       items[i - 1].weight], K[i - 1, w]);
                   else
                       K[i, w] = K[i - 1, w];
               }
           }
           totalVal = K[n, W];
           R_Companies(n, W, K, items);
       }
       private long max(long a, long b) { return (a > b) ? a : b; }
   }
```

توابع

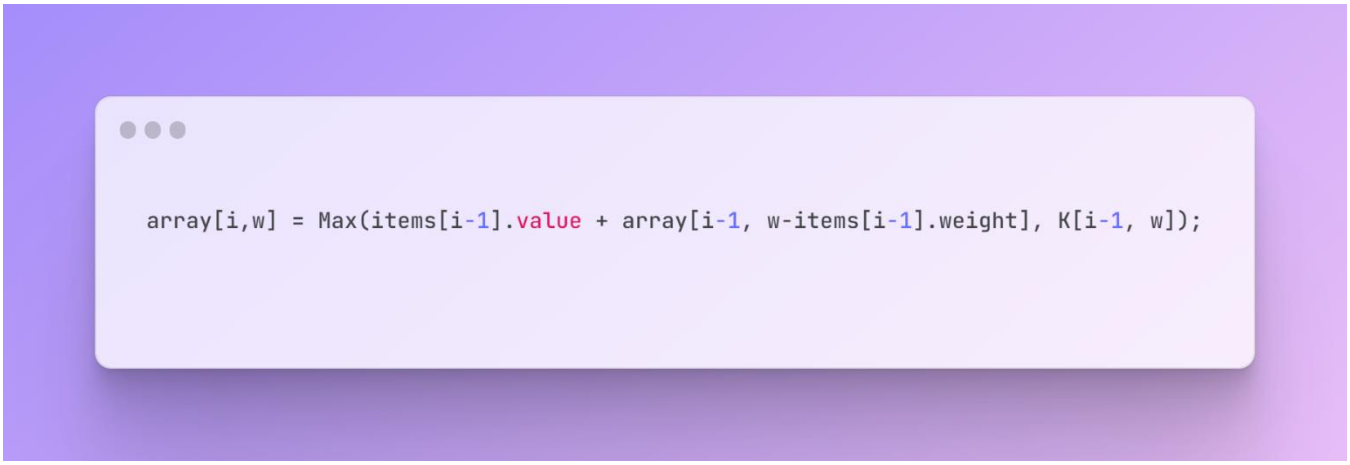
Function -> R_Companies:

لیست شرکت ها را خروجی میدهد

Function ->knapsack_Dynamic:

تابع اصلی برای محاسبه کوله پشتی بر اساس الگوریتم Dynamic

محاسبه هر درایه آرایه



```
array[i,w] = Max(items[i-1].value + array[i-1, w-items[i-1].weight], K[i-1, w]);
```

Time Complexity : O(N*W)

Space Complexity : O(N*W)

اگر W زیاد باشد میواند هزینه زیادی به جا بزارد

الگوریتم Dynamic Optimized

روش 2

تعداد درایه های آرایه کاهش پیدا کرده است

```
public class Dynamic_Algorithm
   {
       public double totalVal = 0;
       public void knapSack_Dynamic_optimized(item[] items, long W)
       {
           long n = items.Length;
           double[] k = new double[W + 1];
           for (long i = 1; i < n + 1; i++)          //Time Complexity  O(N*W)
           {
               for (long w = W; w >= 0; w--)
               {
                   if (items[i - 1].weight <= w)
                   {
                       k[w] = Math.Max(k[w], k[w - items[i - 1].weight]+ items[i -
                       1].value);
                   }
               }
           }
           totalVal = k[W];
       }
   }
```

Time Complexity : O(N*W)

Space Complexity :O(W)

الگوریتم BackTracking

این الگوریتم تمام حالت های ممکن را برررسی میکند و هر کدام که بالا ترین سود را دهد مشخص میکند

```
public class BackTracking_Algorithm
    {
        public long max(long a, long b) { return (a > b) ? a : b; }
        public double totalVal = 0;
        public List<long> company_list = new List<long>();
        public long recurison( item[] items, double W,long n)
        {
            if (n == 0 || W == 0)
                return 0;
            if (items[n - 1].weight > W)
                return recurison( items,W, n - 1);
            else
                return max(items[n - 1].value
                       + recurison( items, W - items[n - 1].weight,   n - 1),
                    recurison( items,W, n - 1));
        }
        public void knapSack_BackTracking(item[] items, double W, long n)
        {
            totalVal= recurison(items,W, n);
        }
    }
```

Time Complexity : O(2^n)

Space Complexity : O(n) -> just for stack

همچنین میتوان با استفاده از الگوریتم branch and bound پیچیدگی زمانی را کاهش داد (با محاسبه boundدر هر node)

$$bound = \underbrace{\left(profit + \sum_{j=i+1}^{k-1} p_j\right)}_{\text{Profit from first } k-1 \text{ items taken}} + \underbrace{(W - totweight)}_{\text{Capacity available for } k\text{th item}} \times \underbrace{\frac{p_k}{w_k}}_{\text{Profit per unit weight for } k\text{th item}}.$$