Liu Mihai-George 262 CTi

Liss

28/01/2022

Examen SDA

2) a)
```
void recur(int n, int K=0)
{
  static int perm[100];
  static int prec[100] = {0};
  if (K==n)
  { for (int i=0; i<n; i++)
      cout << perm[i] << " ";
    cout << "\n";
    return;
  }
  for(int i=1; i<=n; i++)
    if(!prec[i])
    { prec[i]=1;
      perm[K]=i;
      recur(n, K+1);
      prec[i]=0;
    }
}
```

b)
```
int *shuffle(int *a, int *b, int size, int *c)
{ for(int i=0; i<2*size; i++)
    c[i]=a[b[i]];
  return c;
}
```

c. Asumând complexitatea timp pentru algoritmul de generare a tuturor permutărilor fi O(n!) și complexitatea algoritmului de shuffle O(n), atunci acest algoritm de sortare va avea complex. O(n * n!) = [illegible]. Acest algoritm este departe de optim, se pot folosi algoritmi mai optimi d.p.d.v al spațiului și timpului, ex: quicksort.

1. ~~sort(a);~~

```
int binary_search(int a[], int target, int size)
  { int pivot, left=0, right=size-1;
    while (left <= right)
      { pivot = left+(right-left)/2;
        if (a[pivot]==target)
            return pivot;
        if (target < a[pivot])
            right=pivot-1;
        else left=pivot+1;
      }
    return 0;
  }

int main()
{ int a[100]; int size=100; int found=0;
  sort(a);
for(int i=0; i<=size; i++)
    if(binary_search(a, a[i]*a[i]*a[i], size)!=0)
            found=1;
if(found)
  cout<< "Not cube-repetition-free";
else
  cout<< "Cube-repetition-free";
}
```

sortare - O(n log n)
binary-search - (O log n)
de n ori -> O(n log n)

4) b. $x_{i,j} = (b_i - b_j)/(a_j - a_i)$

a. sort(f, functie_comparare)

```
int maxb=0;
for(i=n; i; i--)
  { if(f[i].b <= maxb)
      { f[i].a=-1;
        f[i].b=-1;
      }
    else
      maxb=f[i].b;
  }

for(i=0; i<size; i++)
  { if(f[i].a != -1 && f[i].b != -1)
      vec[i].a = f[i].a;
      vec[i].b = f[i].b;
  }
```

c. $\begin{cases} a_k > a_j > a_i \\ x_{ik} < x_{jk} \end{cases}$ => după pct $x_{i,j}$, ik f_k are valoarea mai mare față de f_i respectiv f_j

=> după x_{ik} se folosește valoarea lui f_k cât f_j nu este folosit niciodată

=> f_j poate fi eliminat

d. După preprocesarea folosind codul de la a) și c) se poate construi un arbore binar de intervale care după $x_{i,i+1}$ din noul vector obținut de la c) și prin parcurgerea acestui arbore în $O(\log n)$ se află funcția care calculează valoarea maximă pt x-ul respectiv.

3) a. Un binary search tree după fiecare operație asupra lui schimbăm subtree-urile astfel încât pentru fiecare nod, numărul de noduri din subtree-ul din stânga ~~minus~~ minus nr. de noduri din subtree-ul din dr, în modul, să fie mai mic sau egal cu 1. Parcurgerea acestuia se execută în $O(\log n)$;
Nodurile din partea stângă din subtree-ul unui nod sunt mai mici decât nodul respectiv, iar pentru partea dreaptă, mai mari.

b. int x = root.dreapta.order
root.dreapta.order -= root.order + 1 (shift la stânga)
root.order = x

int x = root.stânga.order
root.stânga.order -= root.order + 1 (shift la dreapta)
root.order = x

4/4