Classificação por Particionamento

O método de particionamento é um caso de aplicação do princípio da divisão e conquista: classificar dois vetores de tamanho n/2 é mais fácil que classificar um vetor de tamanho n.

O método basicamente se resume em:

i. particionar o vetor;
ii. classificar cada partição.

UNIVASF

Classificação por Particionamento - quicksort

Um processo que trabalha com base nessa proposta é o *quicksort*, assim denominado por seu inventor, C. A. R. Hoare.

É um processo *in situ* e seu caso trivial é a partição de tamanho 1.

Pode ser visto, como uma melhoria do método das trocas, buscando reposicionar primeiro as chaves mais distantes do seu lugar final.

UNIVASF

Classificação por Particionamento - quicksort

Funda-se em separar o vetor em duas partições, delimitadas por uma certa chave, dita *pivô*, de forma que numa das partições estejam todas as chaves menores ou iguais ao pivô e na outra todas as chaves maiores que ele. Em seguida, reclassifica-se cada partição.

Para uma melhor compreensão vamos analisar um exemplo (consideraremos o primeiro elemento da partição como o pivô, com o objetivo de facilitar o processo).

Classificação por Particionamento - quicksort

Se um vetor inicial for dado como:

25 57 48 37 12 92 86 33

e o pivô (25) for colocado na posição correta o vetor resultante será:

12 25 57 48 37 92 86 33

agora, o problema inicial foi decomposto na classificação de dois subvetores:

(12) e (57 48 37 92 86 33)

O primeiro já está classificado, uma vez que tem apenas um elemento. O processo se repete para classificar o segundo subvetor.

Classificação por Particionamento - quicksort

Agora, o vetor pode ser visualizado como:

12 25 (57 48 37 92 86 33)

onde os parênteses encerram os subvetores que ainda serão classificados. Repetindo o processo sobre o subvetor a ordenar teremos:

12 25 (48 37 33) 57 (92 86)

e as aplicações posteriores resultam em:

12 25 (37 33) 48 57 (92 86)

12 25 (33) 37 48 57 (92 86)

12 25 33 37 48 57 (92 86)

12 25 33 37 48 57 (86) 92

12 25 33 37 48 57 86 92

Classificação por Particionamento - quicksort

Os observadores mais atentos devem ter percebido que o quicksort pode ser melhor compreendido definido recursivamente.

É interessante observar que no método apresentado os elementos após o particionamento aparecem na mesma ordem relativa em que apareciam no vetor original. Entretanto, este método de particionamento é ineficiente de se implementar. Veremos agora um método eficiente de se implementar o particionamento.

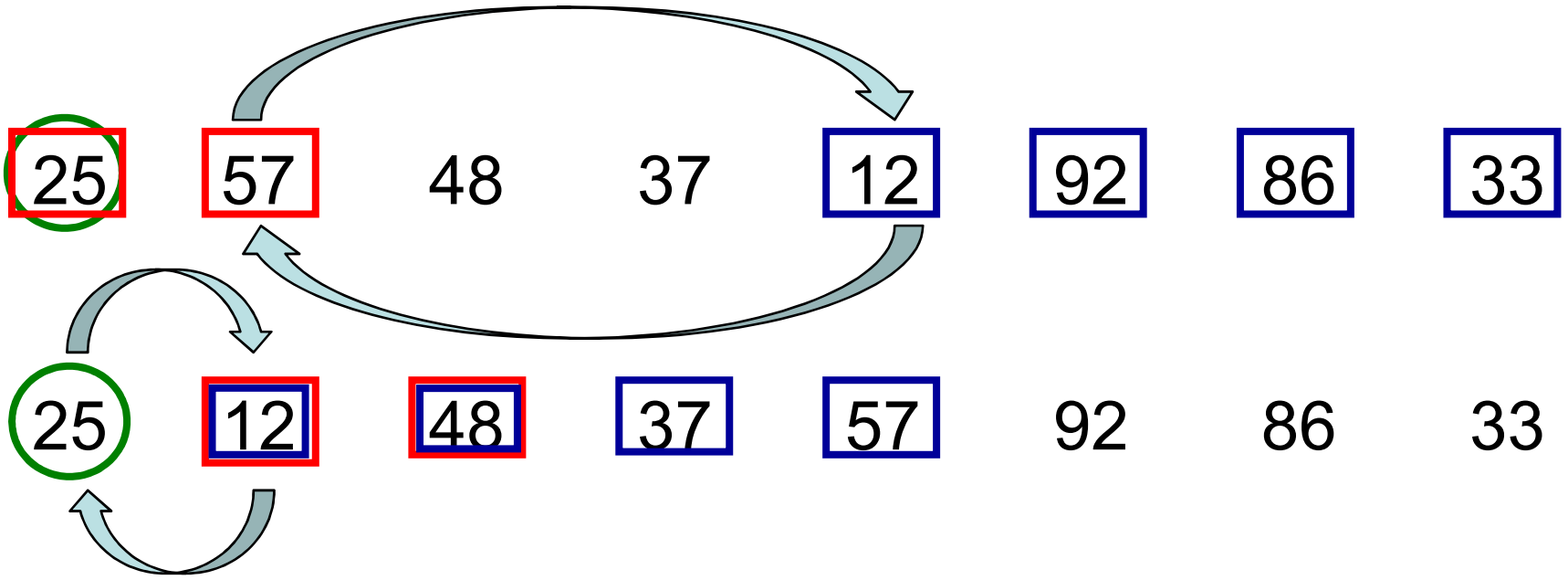
UNIVASF

Classificação por Particionamento - quicksort

Para tanto, com um cursor, digamos “esq”, percorre-se o vetor da esquerda para a direita, até se localizar um elemento maior que o pivô. Com outro cursor, “dir”, percorre-se o vetor da direita para a esquerda, até se encontrar um elemento menor ou igual ao pivô, esses valores são intercambiados caso “dir” seja maior que “esq”. Repete-se este processo até os setores perscrutados esgotarem o vetor, o que se pode detectar pelo encontro dos cursores, ou seja, quando esq > dir. Neste momento, troca-se o valor do pivô pelo valor do elemento indexado por dir que é dito “ponto de partição”. O qual delimitará as partições a serem reclassificadas.

Classificação por Particionamento - quicksort

Este processo pode ser observado no exemplo:



Legenda: Pivô V[esq] V[dir]

Classificação por Particionamento - quicksort

Exercício:

Com base no que foi discutido implemente a função particionar. A qual recebe um vetor, os índices do limite inferior e superior de um subvetor pertencente a este e retorna o índice do ponto de particionamento.

UNIVASF

```
int particionar (int *v, int ii, int is) {
   int esq=ii, dir=is, pivo=v[ii];
   while (esq<dir) {
      while (v[esq]<=pivo && esq<is)
         esq++;
      while (v[dir]>pivo)
         dir--;
      if (esq<dir) {
         int temp;
         temp = v[esq];
         v[esq]=v[dir];
         v[dir]=temp; } }
   v[ii]=v[dir];
   v[dir]=pivo;
   return dir; }
```

Classificação por Particionamento - quicksort

Exercício:

Agora, construa uma função recursiva, em C, que recebe um vetor de inteiros e o número de elementos neste vetor. Esta função deve ordenar o vetor implementando o quicksort.

UNIVASF

```
void quicksort (int *v, int n)
{
   if (n>1)
   {
      int pont_part=particionar(v, 0, n-1);
      quicksort (v, pont_part);
      quicksort (&v[pont_part+1],
       n-1-pont_part);
   }
}
```

Classificação por Particionamento - quicksort

A tarefa de escolher o pivô pode ser executada de forma mais eficiente. A chave ideal para o pivô seria a mediana das chaves, com a qual se teria uma divisão a mais balanceada possível. Só que para determinar a mediana, de forma eficiente, é preciso ordenar o vetor! Como a distribuição das chaves é, em princípio, aleatória, qualquer uma tem a mesma chance de ser mediana. Mas, ao se tomar a primeira, como foi feito anteriormente, corre-se o risco de ser esta a menor (ou a maior) de todas as chaves, tornando praticamente inócuo o trabalho na primeira fase de particionamento (pois se teria uma partição sem nenhum elemento e outra com *n-1* elementos). Uma escolha melhor é a mediana entre a primeira chave, a última e a do meio do vetor.

Classificação por Particionamento - quicksort

Como um exercício, reescreva os algoritmos anteriores, considerando que o pivô não é mais o primeiro elemento mas, sim a mediana entre o primeiro elemento, o último e o elemento do meio do vetor.

UNIVASF

Classificação por Particionamento - quicksort

O desempenho do algoritmo *quicksort* pode ser aquilatado com base nas considerações que seguem. Na fase de particionamento, todos os elementos são comparados com o pivô. Na pior hipótese, todas as chaves seriam trocadas (caso do vetor invertido). Logo, este é o processo **O**(n). Por outro lado, as partições vão diminuindo, e, na melhor hipótese, vão se subdivindo em duas partições de mesmo tamanho (tal ocorre naturalmente quando o vetor está ordenado ou invertido). Nesse caso, gasta-se $\log_2 n$ reclassificações até se chegar às partições de tamanho 1. O melhor desempenho deste processo é então de ordem n log n.

Classificação por Particionamento - quicksort

O pior caso ocorre quando o pivô escolhido é uma chave mínima (ou máxima), pois acarreta uma partição nula e outra de n – 1 elementos. Se isto se repetir em todas as reclassificações, serão necessárias n subsivisões até a conclusão, e o desempenho para o pior caso é **O**(n^2).

A chance de isso ocorrer, na versão melhora do processo que implementamos, é de apenas $1/n^3$ (se houver três chaves mínimas (ou máxima), ocupando exatamente a primeira posição, a intermediária e a última em cada partição).

UNIVASF