|  |
| --- |
|  |
| 여름방학과제 |
|  |
| 10512 김택서 |

**[회사 이름 입력]**

2016.8.18

저자: 김택서

여름방학과제

10512 김택서

**목차**

0.서론

1. 정렬알고리즘

1-1버블

1-2선택

1-3삽입

1-4쉘

1-5퀵

1. 검색알고리즘

2-1순차

2-2이진

1. 문자검색알고리즘

3-1문자열순차검색

3-2KMP알고리즘

1. 후기

-----------------------------------------------------------

0.서론

알고리즘은 프로그래밍을 하는데 사용되는 변수와 배열들에 들어간 정보들을

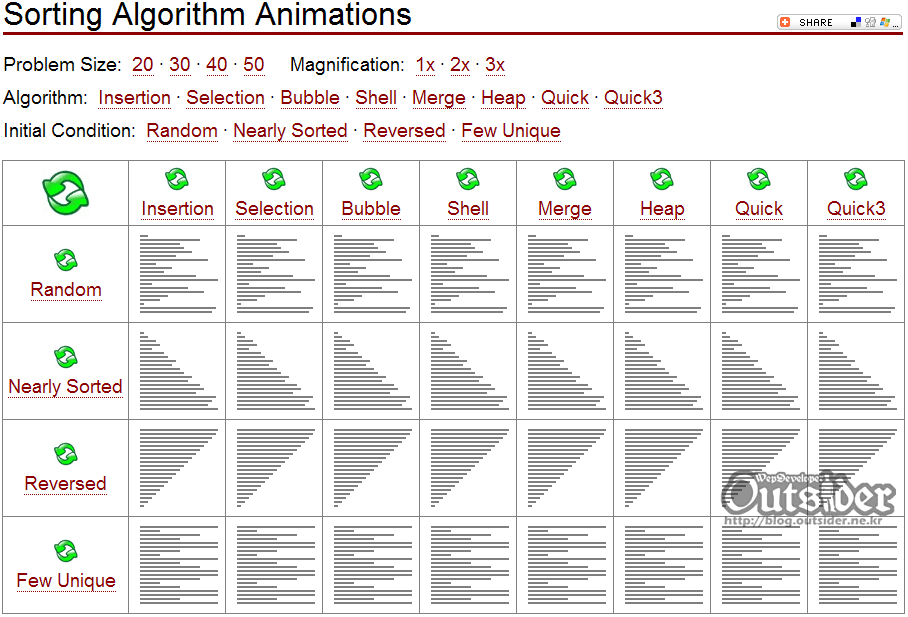
더 효율적으로 관리하고 더 빠르고 정확하며 효율적으로 어떠한 기능을 만들기 위한

또는 어떠한 문제를 해결하기 위한 동작들의 모임으로 이 보고서에서는 여러

알고리즘들 중 정렬알고리즘과 탐색 알고리즘에 관하여 정리하였다 정렬알고리즘은

탐색을 할 때 좀더 빨리 특히나 이진탐색의 경우 엄청난 속도이기 때문에

많은양의 데이터를 다룰 때 중요시 되는 문제이다



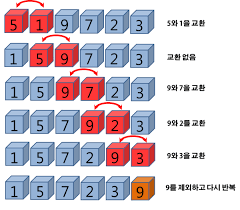
1. 정렬알고리즘

1-1버블정렬

정렬을 할 때 따로 배우지 않아도 자연스럽게 짜게되는 정렬이다

그만큼 간단하게 짤수있고 이해하기도 쉬워서 많이사용된다

정렬방법은 인접한 두 원소를 비교하여 정렬하는것이다



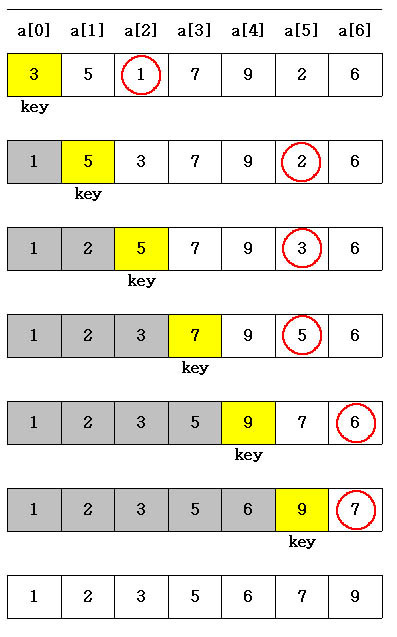
|  |
| --- |
| <코드>버블정렬 |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int main()  {  int i, dev, tmp;  int p[10] = { 0, };//primary  srand((unsigned int)time(NULL));  for (i = 0; i < 10; i++)//내용물 생성  {  p[i] = (rand() % 199) + 1;  printf("%d ", p[i]);  }  for (i = 0; i < 10; i++)//내용물 비교,정렬  {  for (dev = i; dev < 9; dev++)  {  if (p[i] > p[dev])  tmp = p[i], p[i] = p[dev], p[dev] = tmp;  }  }  printf("\n");  for (i = 0; i < 10; i++)//내용물 출력  {  printf("%d ", p[i]);  }  } |
| 풀이 |
| 직접 짰다보니 위키백과에 있는 코드와는 약간 다르다 하지만 인접한 수를 서로 비교하여  정렬한다는 점에서는 같은 버블정렬이라고도 볼 수 있다 |

1-2선택정렬

말그대로 배열의 원소중 최솟값을 선택하여 순서대로 재 배열 하는방식이다

다만 약간 주의할점은 값을 넣을 때 값만 넣는 것이 아니라 기존의 자리에 있던

값과 바꾸어주어야 한다는 점이다.



|  |
| --- |
| <코드>선택정렬 |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int main()  {  int i, dev, min, tmp, tmp2, num = 0;  int p[10] = { 0, };//primary  srand((unsigned int)time(NULL));  for (i = 0; i < 10; i++)//내용물 생성, 분리  {  p[i] = (rand() % 199) + 1;  printf("%d ", p[i]);  }  for (i = 0; i < 9; i++, num = 0)//내용물 정렬  {  min = p[i];  for (dev = i + 1; dev < 10; dev++)  {  if (min > p[dev])  {  min = p[dev];  tmp = dev;  num++;  }  }  if (num)  tmp2 = p[i], p[i] = min, p[tmp] = tmp2;  }  printf("\n");  for (i = 0; i < 10; i++)//내용물 출력  {  printf("%d ", p[i]);  }  } |
| 풀이 |
| 물론 깔끔한 소스는 아니다 하지만 최솟값을 min에다가 넣고 dev로 위치를 저장한다음 num으로 변경이 되었는지를 확인하면서 선택정렬의 틀을 갖추었다고 볼 수 있다. |

1-3삽입정렬

배열의 원소들을 정렬된 배열과 비교하여 순서에 맞는 위치에 넣는것이다

처음 정렬된 배열이란 개념을 이해하기 약간 어려웠는데여기서 정렬되었다는 의미는

마치 기준과도 같은것이었다 배열을 크게 정렬된 부분과 정렬되지 않은 부분으로 나누어

정렬되지 않은 부분을 정렬된 부분에 정렬시켜 넣음으로써 하나씩 증가시켜 모두 졍렬시키는

것이다 말로도 이해하기 어려울수 있지만 사실 일상에서 사람들이 어떠한 기준(ex. 키순

으로 줄을 세울 때 가장 많이 쓰이는 방법이다 먼저 사람들을 모두 불러모으고 한명씩

부르며 기존의 서있던 사람과의 키를 비교하여 줄을 세우는 것처럼 말이다



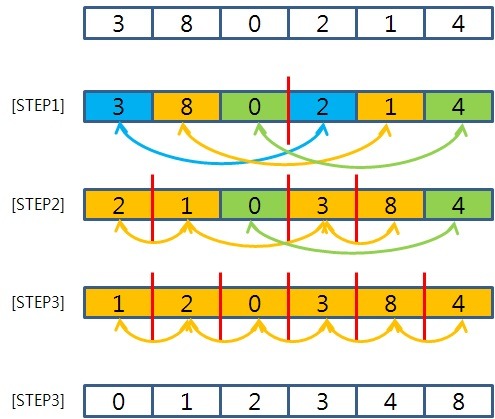
|  |
| --- |
| <코드>삽입정렬 |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int main()  {  int i, j, tmp;  int p[10] = { 0, };//primary  srand((unsigned int)time(NULL));  for (i = 0; i < 10; i++)//내용물 생성  {  p[i] = (rand() % 199) + 1;  printf("%d ", p[i]);  }  for (i = 1; i < 10; i++)//내용물 정렬  {  tmp = p[(j = i)];  while (--j >= 0 && tmp < p[j])  p[j + 1] = p[j];  p[j + 1] = tmp;  }  printf("\n");  for (i = 0; i < 10; i++)//내용물 출력  {  printf("%d ", p[i]);  }  } |
| 풀이 |
| 포문과 와일문으로 앞에서부터 하나하나씩 원소들의 크기를 비교하며  적당한 위치가 될때까지 값을 뒤로 하나씩 밀고 원소를 삽입하는 코드이다 |

1-4쉘정렬

쉘정렬은 원소들을 여러 묶음으로 나누어 삽입정렬을 사용하여 정렬하는 방식을 반복하는

것이다. 삽입정렬은 미리 정렬이 되어있을 때 효율이 더 높아지는데 이를 이용하여

몇몇 부분만(묶음만)을 미리 정렬하여 차근차근 k의 값을 줄여나가면서 모두정렬하게된다



|  |
| --- |
| <코드>쉘정렬 |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int main()  {  int i, j, tmp, k = 10, dev = 0;  int p[10] = { 0, };//primary  srand((unsigned int)time(NULL));  for (i = 0; i < 10; i++)//내용물 생성  {  p[i] = (rand() % 199) + 1;  printf("%d ", p[i]);  }  for (; k != 1;)  {  k = k / 3 + 1; // 간격의 크기  printf("\n간격 k = %d", k);  for (dev = 0; dev < k; dev++)  {  for (i = dev + k; i < 10; i += k)  {  tmp = p[(j = i)];  while (--j >= 0 && tmp < p[j])  p[j + 1] = p[j];  p[j + 1] = tmp;  }  }  printf("\n");  for (i = 0; i < 10; i++)//내용물 출력  printf("%d ", p[i]);  }  } |
| 풀이 |
| 묶음을 얼마나 많이 묶는냐에 따라 원소의 위치가 어떻게 달라지는지도 중요하기 때문에  간격의 크기가 바귈때마다 출력이 된다 |

1-5퀵정렬

피벗을 기준으로 좌우분할하여 좌우 리스틑가 1또는0이될때까지 반복한다

|  |
| --- |
| <코드>퀵정렬 |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  void quickSort(int arr[], int left, int right) {  int i = left;  int j = right;  int pivot = arr[(left + right) / 2];  while (i < j) {  while (arr[i] < pivot) i++;  while (arr[j] > pivot) j--;  int tmp;  tmp = arr[i];  arr[i] = arr[j];  arr[j] = tmp;  }  if (i < right) quickSort(arr, i + 1, right);  if (left < j) quickSort(arr, left, j - 1);  }  int main()  {  int i, dev, tmp;  int p[10] = { 0, };//primary  srand((unsigned int)time(NULL));  for (i = 0; i < 10; i++)//내용물 생성  {  p[i] = (rand() % 199) + 1;  printf("%d ", p[i]);  }  quickSort(p, 0, 9);  printf("\n");  for (i = 0; i < 10; i++)//내용물 출력  {  printf("%d ", p[i]);  }  } |
| 풀이 |
| 혼자 구현하기에는 어려워서 위키의 도움을 받았다  함수에서는 배열과 좌우 사실 그냥 배열의 길이로 받아도 될듯하지만  밑의 조건문에서 복잡해질것같아 놔두었고 피벗은 기준역할을하여  좌우로 분할을하고 재귀를반복한다 |

1-6나만의 정렬

기수정렬과 병합정렬을 잘못보고 만든 정렬이다

이 정렬에서는 하나의 기준을 두고 배열을 분할하여

분할된 배열에서 정렬을 하고 다시 합치는 정렬이다

정확히는 그저 버블정렬을 나누어서 한것이지만 내가 만든 정렬이기도 하고

그냥 없애기에는 아깝기에 추가적으로 작성한다

|  |
| --- |
| <코드> 나만의정렬 |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int main()  {  int i, dev, tmp;  int fc = 0, mc = 0, lc = 0;  int p[10] = { 0, };//primary  int f[10];//first  int m[10];//middle  int l[10];//last  srand((unsigned int)time(NULL));  for (i = 0; i < 10; i++)//내용물 생성, 분리  {  m[i] \*= -1, f[i] \*= -1, l[i] \*= -1;  p[i] = (rand() % 199) + 1;  printf("%d ", p[i]);  if (p[i] / 100)  f[fc++] = p[i];  else if (p[i] / 10)  m[mc++] = p[i];  else  l[lc++] = p[i];  if (i == 9)  break;  }  for (i = 0; i < 9; i++)//정렬  {  for (dev = i; dev < 9; dev++)  {  if (f[i] > f[dev])  tmp = f[i], f[i] = f[dev], f[dev] = tmp;  if (m[i] > m[dev])  tmp = m[i], m[i] = m[dev], m[dev] = tmp;  if (l[i] > l[dev])  tmp = l[i], l[i] = l[dev], l[dev] = tmp;  }  }  printf("\n");  if (1)//재조합(병합 출력  {  for (i = 0; i < lc; i++)  p[i] = l[i];  for (i = lc, dev = 0; i < mc + lc; i++, dev++)  p[i] = m[dev];  for (i = mc + 1, dev = 0; i < fc + mc + lc; i++, dev++)  p[i] = f[dev];  for (dev = 0; dev < 10; dev++)  printf("%d ", p[dev]);  }  } |
| 풀이 |
| 사실 단점밖에 보이지 않는다 버블정렬에 비하여 메모리를 더 먹을 뿐더러  속도도 떨어질것이다 |

1. 검색알고리즘

2-1순차검색

원소들이 정렬되어있지않아 처음부터검색해야 할 때 쓰이는 알고리즘으로 단순하지만 비효율적인 부분이 있다

|  |
| --- |
| <코드>순차탐색 |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int main()  {  int i, fin, tmp = 0;  int p[10] = { 0, };//primary  srand((unsigned int)time(NULL));  for (i = 0; i < 10; i++)//내용물 생성  {  p[i] = (rand() % 199) + 1;  printf("%d ", p[i]);  }  printf("찾으려는 값 입력\n");  scanf("%d", &fin);  for (i = 0; i < 10; i++)//내용물 출력  {  if (p[i] == fin)  {  printf("%d번째에 일치하는 %d값이 있습니다", i, fin);  tmp++; break;  }  }  if (tmp == 0)  printf("값이 없습니다");  } |
| 풀이 |
| 반복문으로 배열의 앞에서부터 비교하는 순차검색의 예시이다 |

2-2이진검색

데이터들이 정렬되어 있을 때 사용할 수 있는 알고리즘으로 데이터를 반씩 잘라가며 비교를

하기 때문에 매우 빠른속도로 값을 찾을 수 있다

|  |
| --- |
| <코드>이진탐색 |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int BinarySearch(int dataArr[], int size, int findData)//  {  int low = 0, high = size - 1, mid;  while (low <= high)  {  mid = (low + high) / 2;//중간계산  if (dataArr[mid] > findData) high = mid - 1;//클경우 범위변경  else if (dataArr[mid] < findData) low = mid + 1;//작을경우 범위변경  else return mid;//찾았을때  }  return -1;  }  int main()  {  int i, j, fin, tmp = 0;  int p[10] = { 0, };//primary  srand((unsigned int)time(NULL));  for (i = 0; i < 10; i++)//내용물 생성  {  p[i] = (rand() % 199) + 1;  printf("%d ", p[i]);  }  for (i = 0; i < 10; i++)//내용물 비교,정렬  {  for (j = 0; j < 9; j++)  {  if (p[j] > p[j+1])  tmp = p[j], p[j] = p[j+1], p[j+1] = tmp;  }  }  printf("\n");  for (i = 0; i < 10; i++)  printf("%d ", p[i]);  printf("찾으려는 값 입력\n");  scanf("%d", &fin);  tmp = BinarySearch(p, 10, fin);  if (tmp == -1)  printf("값이 없습니다");  else  printf("%d번째에서 %d발견", tmp + 1, fin);  } |
| 풀이 |
| 탐색시에 중간보다 값이 큰경우는 중간+1부터 끝까지를 범위로삼고 작을때는  처음부터 중간-1까지로 범위를만든다음 매 시작마다 새로운 중간을 만들어 정렬한다 |

1. 문자검색알고리즘

3-1순차검색알고리즘

위의 순차검색 알고리즘과 비슷한 원리이다

|  |
| --- |
| <코드>순차적 검색알고리즘 |
| #include <stdio.h>  #include <stdlib.h>  int main()  {  int i, j, dev=0;  char p[13] = "abaaabbab";  char fin[6] = { 0, };  printf("%s\n 찾을 문자열을 입력하세요(세글자)", p);  gets(fin);  for (i = 0; i < 10; i++)  {  for (j = 0; j < 3; j++)  {  if (p[i] == fin[j])  if (p[i + 1] == fin[j + 1])  if (p[i + 2] == fin[j + 2])  {  printf("%d번째부터 %s가 있습니다", i + 1, fin);  return 0;  }  }  }  } |
| 풀이 |
| 약간 무식한방법으로 배열과 입력된 문자열을 비교했지만  그래도 문자열을 처음부터 순차적으로 찾는다는 것은 같다  오버플로우를 방지하기위해 배열을3칸씩늘렸고 중복되어 여러 번출력되는 부분이 없도록  하기위해 바로 함수를 종료시켰다 |

3-2KMP알고리즘

KMP은 패턴으로 문자열을 비교하고 이동시키기 때문에

찾을 문자열의 패턴을 분석하고 찾고자하는배열에서 일치하지 않을경우

다음 패턴으로 넘어가는 방식이다 이때 패턴은 찾고싶은문장에서 접두사와

접미사로 나누어 지는데 둘이 같은 모습이어야 한다 때문에 접두사쪽에서

패턴이 맞지 앉는다 해도 바로 접미사로 가면 되는것이다

|  |
| --- |
| <코드> |
| #include <iostream>  #include <string>  #include <cstdio>  #include <vector>  using namespace std;  vector<int> getPi(string p) {  int m = (int)p.size(), j = 0;  vector<int> pi(m, 0);  for (int i = 1; i< m; i++) {  while (j > 0 && p[i] != p[j])  j = pi[j - 1];  if (p[i] == p[j])  pi[i] = ++j;  }  return pi;  }  vector<int> kmp(string s, string p) {  vector<int> ans;  auto pi = getPi(p);  int n = (int)s.size(), m = (int)p.size(), j = 0;  for (int i = 0; i < n; i++) {  while (j>0 && s[i] != p[j])  j = pi[j - 1];  if (s[i] == p[j]) {  if (j == m - 1) {  ans.push\_back(i - m + 1);  j = pi[j];  }  else {  j++;  }  }  }  return ans;  }  int main() {  string s, p;  getline(cin, s);  getline(cin, p);  auto matched = kmp(s, p);  printf("%d\n", (int)matched.size());  for (auto i : matched)  printf("%d ", i + 1);  return 0;  } |
| 풀이 |
| 위 소스는 직접 짠 것이 아닌 구글링으로 찾은 소스이다  cpp문법이라 모두 알아볼수는 없지만 KMP에 대하여 알아본것만으로는  getPi에서는 패턴을 분석하고 KMP함수에서 문자를 찾을 것 같다 |

1. 후기

아무래도 시간에 못이겨 바삐한게 가장 큰 오류인 것 같다

해야지 해야지 하면서도 미룬탓이다.여런 알고리즘들과

새로운 문법들을 보면서 아직도 배울 것은 많다는 생각도들었다

코드를 보고 설명을보고 동영상을 보고 친구들에게묻고

구글링을해봐도 이해가 안가는 것들도 있지만 지금은

그냥 새로운것이니 두번 세번보면 이해가 될것같기도 하다