숭실대학교 Soongsil University

알고리즘 2021 보고서

보고서 제출서약서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
   1.1. 나는 동료의 보고서를 베끼지 않았습니다.
   1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히 인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)
3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.
4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

과목	알고리즘(나) 2021
과제명	과제 7
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20163340 강원경 (출석번호 201번)
제출일	2021년 11월 11일

차 례

1장. 문제 11 수행 결과

문제 11은 체스판의 크기가 8×8인 n-Queens problem을 Monte Carlo 예측 수행을 20번 반복하고 그 평균치를 구하는 문제이다.

이 문제를 수행하기 위해 2개의 함수를 작성했는데, 첫 번째 함수는 유망(Promising)한지를 검사하는 함수이고, 나머지 하나는 n-Queens Problem에 대해 Monte Carlo 알고리즘을 수행하는 함수이다.

```
55 // n-queens에서 현재 노드가 유망한지 검사하는 함수
56 // i: 현재 탐색중인 행 번호, col: 현재까지의 탐색 결과가 저장된 배열
57 int queen_promising(int i, int* col){
58     int k=1, chk=1;
59 
60     while(k<i && chk){
61         if(*(col+i)==*(col+k) || (*(col+i)-*(col+k))==i-k)
62             return chk=0;
63         k++;
64     }
65     return chk;
66 }
```

queen_promising 함수는 해당 노드가 유망한지 검사하는 함수로, 현재 위치에서 8방향에 다른 퀸이 놓여있는지 검사하고 없다면 1, 있다면 0이 출력된다.

```
68 // n-queens 알고리즘에 대한 Monte Carlo 분석을 수행하는 함수
69 // n: 체스판 크기
70 int estimate_n_queens(int n){
71     int m=1, mprod=1; // m: 자식 노드 개수, mprod: 현재 행에서의 예상 이웃노드 개수
72     int numnodes=1, i=0; // numnodes: 전체 예상 탐색 노드 개수, i: 현재 탐색중인 행 번호
73     int col[n+1]; // 현재까지 탐색한 결과가 저장된 배열. (ex. col[1]=1번째 행은 몇 번째 열에 퀸을 놓으면 되는지)
74     int promisingChild, j; // promisingChild: 유망한 자식 노드에 대한 집합을 bitmask 형태로 표현. j: 인덱스
75 
76     while(m!=0 && i!=n){
77         mprod*=m;
78         numnodes+=(mprod*n);
79         i++; // 행 번호 증가
80         m=0;
81         promisingChild=0;
82         for(j=1; j<=n;j++){ // 열을 1개씩 증가시켜가며 탐색
83             col[i]=j;
84             if(queen_promising(i, col)){ // 유망한 노드라면
85                 m++; // 유망한 노드 개수 증가
86                 promisingChild|=(1<<j); // 유망한 자식 집합에 j열을 넣음
87             }
88         }
89         if(m!=0){ // 유망한 노드가 1개라도 있다면
90             // 유망한 자식노드가 선택될 때까지 반복적으로 랜덤픽 진행.
91             while(1){
92                 j=((int)rand()%n)+1;
93                 if(promisingChild&(1<<j))
94                     break;
95             }
96             col[i]=j; // 임의로 지정한 j번째 열을 i행의 탐색 결과로 넣음
97         }
98     }
99     return numnodes;
100 }
```

estimate promising 함수는 주어진 체스판 크기 n에 대해 n-Queens problem을 수행하는 함수로, 유망한지 여부는 앞서 소개한 queen_promising 함수를 이용해 검사한다.

```c
int main() {
    printf("[문제 11] n-queens를 20번 수행할 때의 평균 복잡도를 Monte Carlo를 이용해 구하기\n");
    int queenMonteSum=0, monteResult[20];
    srand((unsigned int)time(NULL));
    for(int i=0;i<20;i++){
        monteResult[i]=estimate_n_queens(8);
        queenMonteSum+=monteResult[i];
    }
    printf("--> n-queens 20번 수행 시 검사하는 노드 수의 평균: %d\n\n", queenMonteSum/20);
```

위에서 설명한 2개의 함수를 이용해 main에서 20번 수행하게 되고, 그 수행 결과는 아래와 같다.

```
[문제 11] n-queens를 20번 수행할 때의 평균 복잡도를 Monte Carlo를 이용해 구하기
--> n-queens 20번 수행 시 검사하는 노드 수의 평균: 64847
```

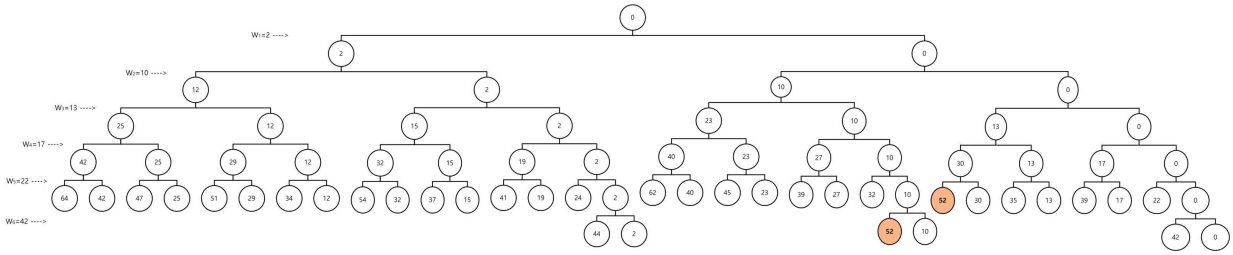
이 수행 결과값은 Monte Carlo 예측의 특성상 매 실행마다 노드가 랜덤하게 선택되고 그에 따라 결과값에도 변동하기 때문에 평균적인 값은 변화할 수 있다. 본인이 여러 번 수행 했을 때 값이 대부분 6만~10만대 정도의 범위 안에서 출력되었다.

2장. 문제 13 수행 결과

(문제 13은 본래 프로그램을 구현하는 문제로 이해하고 과제를 수행하였으나, mini 커뮤니티의 질의응답을 통해 수기로 수행하는 문제로 확인되어 그 기준에 맞게 과제를 수행하였습니다.)

문제 2는 주어진 데이터{2,10,13,17,22,42}에 대해서 합이 52가 되는 경우를 모두 보이고, 그 수행 단계를 서술하는 문제이다.

각 단계별로 보이는 것은 상태 트리를 이용하는 것이 가장 효율적이므로 이 문제에 대한 상태트리를 그렸다.



(첨부한 그림 파일은 크기가 너무 커서 글씨가 보이지 않아,
확대해서 보실 수 있도록 제출한 압축파일 안에 별도의 이미지로 첨부하였습니다.)

위의 트리에 들어있는 값은 해당 노드까지의 선택한 데이터들의 합(weight)이고, 왼쪽 자식 노드는 부모 노드 레벨의 인덱스에 해당하는 데이터를 포함시켰을 때의 weight, 오른쪽 자식 노드는 그 데이터를 포함하지 않았을 때의 weight이다.

또한, 트리에서 자식 노드를 단 1개도 가지지 않는 노드는 그 노드가 마지막 레벨의 노드이거나, 자식 노드 중에 어느 하나도 유망한 노드가 없는 경우, 또는 이미 찾으려는 목표값에 도달한 경우이다.

3장. 문제 14 수행 결과

문제 14는 앞서 수기로 풀었던 sum-of-subsets 문제를 직접 코드로 구현하는 문제이다.
이 문제를 해결하는데 사용된 함수는 아래의 2개이다.

```
102 // sum of subsets를 수행하는 함수
103 // i: 탐색하는 행의 번호 (0=루트), weight: 현재까지 얻은 이익의 총 합
104 // total: 앞으로 남은 데이터들을 모두 더할 때 얻을 수 있는 이익의 최대치
105 // W: 탐색 목표값, arr: 입력 데이터가 들어있는 배열
106 // include: 현재 노드값을 포함하는지 여부. 포함할 경우 해당 인덱스의 데이터가 들어가고, 포함하지 않을 경우 0이 들어감.
107 void sum_of_subsets(int i, int weight, int total, int W, int* arr, int* include){
108     if(subset_promising(i, weight, total, W, arr)){
109         if(weight==W){ // W값과 동일한 덧셈의 조합을 찾은 경우 출력하는 부분
110             for(int idx=1;idx<=i;idx++){
111                 int cur=*(include+idx);
112                 if(cur){
113                     printf("%d",cur);
114 
115                     if(idx!=i)
116                         printf(" + ");
117                     else printf("\n");
118                 }
119             }
120 
121         }
122         else{ // W값과 동일하지 않은 경우
123             // 현재 노드 값을 포함하는 자식 노드를 탐색
124             include[i+1]=arr[i+1];
125             sum_of_subsets(i+1, weight+arr[i+1], total-arr[i+1], W, arr, include);
126 
127             // 현재 노드 값을 포함하지 않는 자식 노드들을 탐색
128             include[i+1]=0;
129             sum_of_subsets(i+1, weight, total-arr[i+1], W, arr, include);
130         }
131     }
132 }
```

첫 번째 함수는 sum_of_subsets함수로, 이 알고리즘에서 주요한 부분을 수행하는 함수이다. weight, total, W를 이용해 해당 위치의 노드가 유망한지를 검사하고, 유망하다면 계속적으로 탐색을 수행하고 아닐 경우 이전 노드로 되돌아간다.

해당 노드가 유망한지 여부를 검사하는 부분은 두 번째로 설명할 함수에서 수행한다.

```
134 // 현재 노드가 유망한지 검색
135 // i: 유망 여부를 판단하려는 입력 데이터의 인덱스 번호, weight: 현재 노드에 포함된 가치의 합
136 // total: 앞으로 남은 노드들을 모두 더할 때 얻을 수 있는 최대의 가치
137 // W: 탐색 목표값, arr: 입력 데이터가 들어있는 배열
138 int subset_promising(int i, int weight, int total, int W, int* arr){
139     return (weight+total>=W)&&(weight==W||weight+*(arr+i+1)<=W);
140 }
```

subset_promising 함수는 아래의 2가지 조건을 모두 만족하는 경우 유망하다고 판단한다.

1. 현재 얻은 가치의 합과 앞으로 이론상 얻을 수 있는 최대의 이익을 합쳐 목표값보다 같거나 크다.
2. 현재 얻은 가치의 합과 목표값이 같거나, 다음 노드의 값을 더했을 때 목표값 W보다 작거나 같다.

위 두 개의 함수를 이용해 주어진 데이터에 대해 코드를 수행하면 아래와 같은 결과가 출력된다.

```
[문제 14] 2, 10, 13, 17, 22, 42에 대한 Sum-of-Subsets 해결
10 + 42
13 + 17 + 22
```

4장. 문제 17 수행 결과

문제 17은 Monte Carlo 알고리즘을 이용해 앞서 작성했던 sum-of-subsets 문제의 효율성을 예측하는 문제이다. 문제 11에서 작성했던 알고리즘과는 달리 sum-of-subsets 문제에 맞도록 Monte Carlo 알고리즘에 수정해야 할 부분이 있어 새로운 함수를 정의하여 문제를 해결하였다.

```
142 // sum of subset에 대한 Monte Calro 알고리즘
143 // n: 데이터 개수, total: 모든 데이터의 합, W: 찾으려는 합, arr: 데이터가 들어있는 배열
144 int estimate_subsets(int n, int total, int W, int* arr){
145     int m=1, mprod=1; // m: 현재 노드의 자식 중 유망한 노드 개수, mprod: 현재 레벨에서 유망할 것으로 예상되는 노드 수
146     int numnodes=1, i=0; // numnodes: 예상되는 탐색 노드 개수, i: 탐색하려는 자식 노드의 인덱스
147     int remain_tot=total; // remain_tot: 앞으로 얻을 수 있는 최대의 이익
148     int tree[n+1]; // 현재 데이터까지의 탐색 결과 (얻은 이익의 총 합)
149     int promisingChild, j; // promisingChild: 유망한 자식 노드에 대한 집합을 bitmask 형태로 표현. j: 인덱스
150     tree[0]=0;
151     while(m!=0&&i!=n){
152         mprod*=m;
153         numnodes+=(mprod*2);
154         i++; // 자식 노드의 인덱스로 이동.
155         m=0; // 유망한 자식 노드 갯수 초기화
156         promisingChild=0; // 유망한 자식 노드의 집합을 공집합으로 초기화
157         for(j=1; j<=2;j++){ // j==1 : 현재 레벨의 데이터 포함. j==2 : 현재 레벨의 데이터 미포함.
158             tree[i]=tree[i-1]+arr[i]*(2-j);
159             if(subset_promising(i, tree[i], remain_tot-arr[i]*(j-1), W, arr)){
160                 m++;
161                 promisingChild|=(1<<j);
162             }
163         }
164         if(m!=0){ // 유망한 자식 노드가 있다면
165             // 유망한 자식노드가 선택될 때까지 반복적으로 랜덤픽 진행.
166             while(1){
167                 j=((int)rand()%2)+1;
168                 if(promisingChild&(1<<j))
169                     break;
170             }
171             remain_tot-=arr[i]; // 현재 데이터를 포함하지 않는 경우 이론상 최대 이익의 값을 감소시킴.
172             tree[i]=tree[i-1]+arr[i]*(2-j);
173         }
174     }
175 //  printf("%2d | ", tree[i]);
176     return numnodes;
177 }
```

대부분의 알고리즘은 문제 11과 비슷하지만, 탐색해야 할 자식 노드가 2개로 줄었다는 것, 그리고 유망한지 여부를 판단하는 함수와 total의 값에 대한 처리가 필요한 부분이 있었다.

위 알고리즘을 사용하여, 입력 데이터를 문제 13에서의 6개 데이터를 이용해 main 함수에서 20번 수행했고,

```
44      printf("[문제 17] sum-of-subsets를 20번 수행한 평균 복잡도를 Monte Carlo를 이용해 구하기\n");
45      int subsetMonteSum=0;
46      for(int i=0;i<20;i++){
47  //      printf("%d번째 실행: ", i+1);
48          int est=estimate_subsets(n, total, W, arr);
49  //      printf("탐색횟수: %d\n", est);
50          subsetMonteSum+=est;
51      }
52      printf("--> sum-of-subsets 20번 수행 시 검사하는 노드 수의 평균: %d\n", subsetMonteSum/20);
```

평균의 결과는 아래와 같이 출력되었다.

```
[문제 17] sum-of-subsets를 20번 수행한 평균 복잡도를 Monte Carlo를 이용해 구하기
--> sum-of-subsets 20번 수행 시 검사하는 노드 수의 평균: 59
Program ended with exit code: 0
```