Modified Knapsack Problem

Akshit Kumar (EE14B127)

27th November 2016

Abstract

This report contains the code and the algorithm used for the first question of the take home end-semester examination. The objective of the problem is to find a subset of N objects of positive weights $\{w_i\}$that maximizes their sum subject to a different constraint than the original knapsack constraint. Both a dynamic programming and greedy solution is given to the problem in the question

```c
// Including the necessary libraries
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include <string.h>
#include <limits.h>

#define max(a,b) (a > b ? a : b)

typedef struct Cost{
  int weight;
  int count;
}Cost;

Cost dp[2000][2000];

int W;
int weights[2000];
int knapsack_elements_dp[2000];
int knapsack_elements_greedy[2000];
int num_elements = 0;
int compare_function(const void *a, const void *b){
  return ( *(int*)b - *(int*)a);
}

int get_limit(char line[]){
  char *pos;
  int W;
  if((pos = strchr(line,'/n')) != NULL){
    *pos = '\0';
  }
  char *token;
  token = strtok(line," ");
  while(token != NULL){
    sscanf(token,"%d",&W);
    token = strtok(NULL," ");
  }
  return W;
}

int get_weights(char line[]){
  char *pos;
  if((pos = strchr(line,'\n')) != NULL){
    *pos = '\0';
  }
  char *token;
  token = strtok(line," ");
  int num;
  int i = 1;
  weights[i++] = INT_MAX;
  while(token != NULL){
    sscanf(token,"%d",&num);
    weights[i++] = num;
    token = strtok(NULL," ");
  }
  return i-1;
```

```
}

int dp_knapsack_solution(int W, int n){
  for(int j = 0; j <= W; j++){
    dp[0][j].weight = 0;
    dp[0][j].count = 0;
  }
  for(int i = 1; i <= n ;i++){
    for(int j = 0; j <= W; j++){
      if(weights[i] > j){
        dp[i][j] = dp[i-1][j];
      }
      else{
        if(dp[i-1][j-weights[i]].weight + weights[i] >
            dp[i-1][j].weight &&
            dp[i-1][j-weights[i]].weight + weights[i] <=
            (float)j - log(dp[i-1][j-weights[i]].count + 1)){
          dp[i][j].weight = dp[i-1][j-weights[i]].weight +
              weights[i];
          dp[i][j].count = dp[i-1][j-weights[i]].count + 1;
        }
        else{
          dp[i][j] = dp[i-1][j];
        }
      }
    }
  }
  return dp[n][W].weight;
}

int get_knapsack_elements(int W,int n){
  int line = W;
  int i = n;
  int num_elements = 0;
  while(i > 0){
    if(dp[i][line].weight > dp[i-1][line].weight){
      knapsack_elements_dp[num_elements++] = weights[i];
      line = line - weights[i];
      i--;
    }
    else{
      i--;
    }
  }
  return num_elements;
}

int greedy_knapsack_solution(int W,int n){
  qsort(weights,n,sizeof(int),compare_function);
  num_elements = 0;
  int i = 1;
  int sum = 0;
  while(i <= n){
    sum += weights[i];
    num_elements++;
    if((sum <= (float)W - log(num_elements))){
      knapsack_elements_greedy[num_elements] = weights[i];
```

```
    }
    else{
      sum -= weights[i];
      num_elements--;
    }
    i++;
  }
  return sum;
}

void print_dp_matrix(int W, int n){
  for(int i = 0; i <= n; i++){
    for(int j = 0; j <= W; j++){
      printf("%d ", dp[i][j].weight);
    }
    printf("\n");
  }
}

void write_dp_solution_to_file(int W,int n){
  FILE *file = fopen("output1.dat","a");
  fprintf(file,"Dynamic Programming Table : \n");
  for(int i = 0; i <= n; i++){
    for(int j = 0; j <= W; j++){
      fprintf(file,"%d ", dp[i][j].weight);
    }
    fprintf(file,"\n");
  }
  fprintf(file,"\n");
  fclose(file);
}

void print_knapsack_elements_dp(int N){
  for(int i = 0; i < N ; i++){
    printf("%d ",knapsack_elements_dp[i]);
  }
  printf("\n");
}

void print_knapsack_elements_greedy(int N){
  for(int i = 1; i <= N; i++){
    if(knapsack_elements_greedy[i] != 0){
      printf("%d ",knapsack_elements_greedy[i]);
    }
  }
  printf("\n");
}

int main(int argc,char **argv){
  if(argc != 2){
    printf("Usage ./a.out <filename>\n");
    exit(1);
  }
  FILE *file = fopen(argv[1],"r");
  if(file == NULL){
    printf("Unable to open file\n");
    exit(1);
```

```
  }
  char line[2000];
  char line_number = 0;
  int W;
  int n;
  while(fgets(line,sizeof line,file) != NULL){
    if(line[0] == '#' || line[0] == '\n'){
      continue;
    }
    else{
      line_number++;
      if(line_number % 2 != 0){
        W = get_limit(line);
      }
      else if(line_number % 2 == 0){
        n = get_weights(line);
        line_number -= 2;
      }
      if(line_number == 0){
        printf("DP Solution : %d\n",
            dp_knapsack_solution(W,n));
        printf("Elements in the Knapsack due to DP Approach
            : ");
        print_knapsack_elements_dp(get_knapsack_elements(W,n));
        write_dp_solution_to_file(W,n);
        printf("Greedy Solution :
            %d\n",greedy_knapsack_solution(W,n));
        printf("Elements in the Knapsack due to Greedy
            Approach : ");
        print_knapsack_elements_greedy(num_elements);
        printf("--------------------------------------------------------------------------------
      }
    }
  }
  return 0;
}
```

knapsack.c