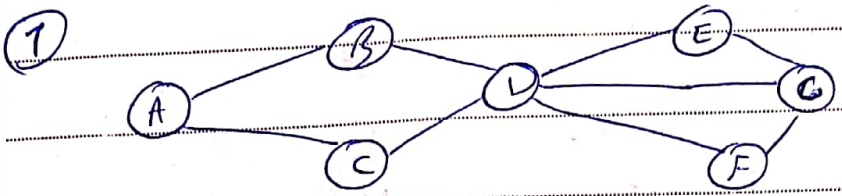
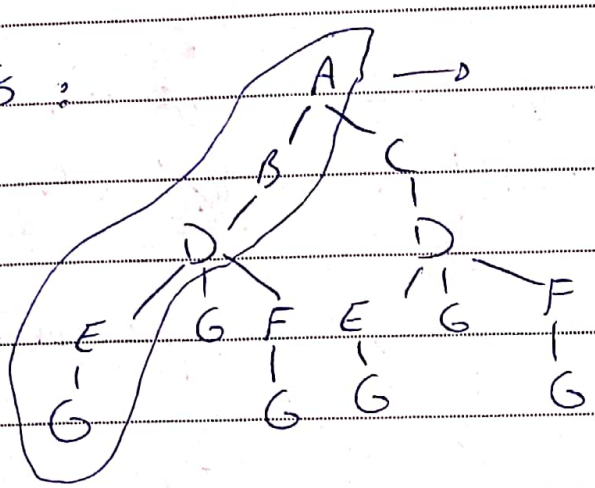
امیرمحمد کیمیایی hw1

Subject:
Year: Month: Date:



DFS :

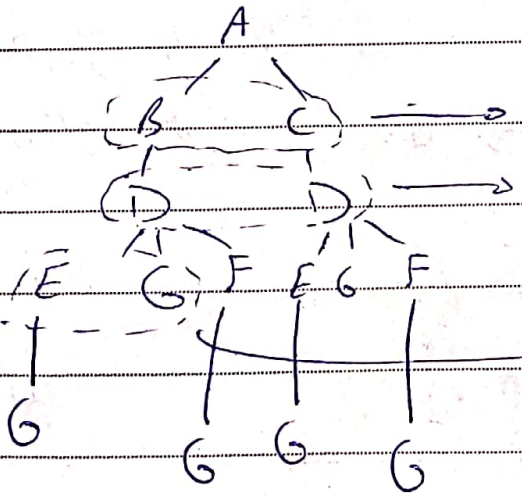


مسیر پیاده شده با استفاده از DFS:

از child سمت چپ هر node شروع می‌کنیم و تا انتها می‌رویم و در اولین جهت چپ به G (جواب نهایی) می‌رسیم

ABDEG

BFS:



در الگوریتم BFS بعد از root اولین سطری که خوانده می‌شود

سپس به سطر بعدی می‌رویم و آنرا می‌خوانیم

سپس به سطر بعدی می‌رویم و در همین جا با رسیدن به هدف یعنی G الگوریتم به جواب نهایی رسید و متوقف می‌شود

ABCDEG

hw1

Subject:
Year: Month: Date:

Sa Su Mo Tu We Th

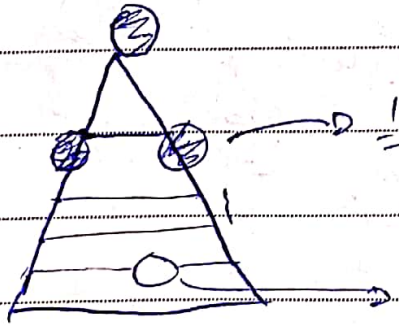
② Iterative Deepening

توضیح روش: می دانیم که در روش DFS , BFS هر دو از لحاظ زمانی و مکانی دارای معایبی هستند.

از ترکیب این دو روش و مزایای این دو در روش Iterative Deffening استفاده می کنیم. یعنی از بهتری DFS در space و از بهتری BFS در time استفاده می کنیم.

ابتدا به این صورت که با استفاده از DFS هر عمق را انتخاب کرده سپس هر عمق را که مانند سطرات از BFS استفاده می کنیم و این روش را تا رسیدن به جواب روی هر عمق امتحان می کنیم.

مشکل این روش این است که در هر عمق، اعماق قبلی را دوباره می سنجد می کند



کاربردها: ① زمانی که عمق مساله را نمی دانیم و حجت و جود روی یک state space بزرگ انجام می شود

② استفاده کمتر از حافظه نسبت به BFS

Subject:
Year: Month: Date: hw1

Sa Su Mo Tu We Th

③ به طور کلی برای حل سوالات پازل (پازل مانند سوال یا همان 15-Puzzle)

روش BFS همواره جواب خواهد داد اما روش DFS به این بستگی دارد که

عمق ما محدود باشد (finite depth)

DFS : $O(b^m)$, BFS : $O(b^{d+1})$

m : حداکثر عمق

d = عمقی که در آن به جواب رسیده‌ایم

برای m و d حالات مختلفی داریم ←

DFS بهتره ← m = d

BFS بهتره ← m > d

BFS ممکنه بهتر باشه ← m = infinite

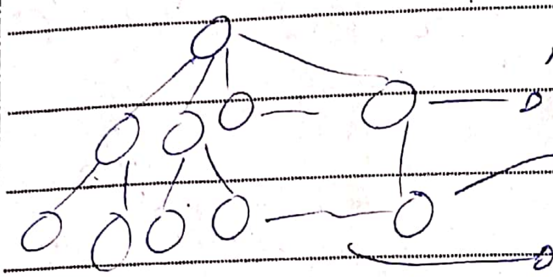
DFS = fail ←

hw1 DFS & BFS

Subject: Year: Month: Date:

الگوریتم و توضیح کد BFS :

می دانیم الگوریتم BFS به این صورت کار می‌کند که هر سطح اسطر را در ابتدا بررسی کرده و سپس به سطح اسطر بعدی می‌رود.



در ابتدا دو متغیر به نام‌های first Location و final location داریم که از تابع آماده در کد Algorithm.py ، مختصات نقطه شروع مار و نقطه‌ای که باید در آن میوه/خوراکی را شکار کند در این دو وارد می‌شود. first location در صف frontier نیز قرار می‌گیرد. بعد از این با توجه به حلقه while ، دو لیست frontier و explored آپدیت خواهند شد. ابتدا راسی به نام shallowest_node داریم که آن را در لیست explored ، اضافه می‌کنیم به عنوان نودی/راسی که بازدید شده سپس همسایه این node را در متغیری به نام neighbor ذخیره می‌کنیم. حال در حلقه for ، ابتدا با استفاده از یک تابع بررسی می‌کنیم که آیا راس neighbor داخل محوطه تعریف شده است یا خیر که اگر نبود به آن سمت نمی‌رویم و اگر بود بازی را ادامه می‌دهیم

hw1 DFS & BFS

Sa Su Mo Tu We Th

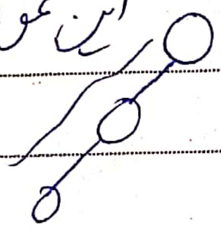
Subject:
Year: Month: Date:

سپس neighbor را در لیست explored به عنوان رأس بازدیدشده قرار می‌دهیم و در frontier هم
می‌گذاریم تا آن را گسترش دهیم اگر مختصات neighbor با مختصات final loc
برابر بود مار به آن سمت می‌رود و آن را انتخاب می‌کند در غیر این صورت None برمی‌گرداند

منبع Github

توضیح کد DFS

ابتدا یک تابع بازگشتی داریم که ← الگوریتم DFS



اگر به جواب رسیدیم path را برگرداند (راه اتمام تابع بازگشتی)

اگر موقعیت فعلی ما قبلاً بازدید شده بود None برگرداند

سپس همانند BFS، neighbors را تشکیل می‌دهیم و در تابع DFS_check (تابع خودم در فایل Algorithm.py)
وضعیت را چک می‌کنیم که اگر بیرون بود [illegible] آن راه را نرود و اگر تو بود بررسی
کند اگر به میوه نرسیدیم تابع بازگشتی را فراخوانی کند و اگر به میوه رسیدیم
path را برگرداند و اتمام دهد